

RADC-TR-89-268
Final Technical Report
December 1989



AD-A219 216

FAULT TOLERANT MULTIPROCESSING AND DEVELOPMENT OF TOOLS

Pennsylvania State University

Sponsored by
Strategic Defense Initiative Office



APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Strategic Defense Initiative Office or the U.S. Government.

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

90 03 09 071

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Services (NTIS) At NTIS it will be releasable to the general public, including foreign nations.

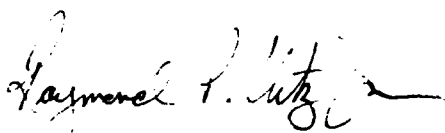
RADC-TR-89-268 has been reviewed and is approved for publication.

APPROVED:



MARK D. FORESTI
Project Engineer

APPROVED:



RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER:



IGOR G. PLONISCH
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COTC) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

FAULT TOLERANT MULTIPROCESSING AND DEVELOPMENT OF TOOLS

Dr. T. Feng
Dr. C. R. Das
Dr. P. T. Hulina
Dr. A. R. Hurson
Dr. S. Pakzad
Dr. M. J. Thazhuthaueetil
Dr. C. Wu
Dr. D. L. Landis
Dr. W. Lin
Dr. J. J. Metzner

Contractor: Pennsylvania State University
Contract Number: F30602-86-C-0005
Effective Date of Contract: 11 December 1985
Contract Expiration Date: 14 November 1988
Short Title of Work: Fault Tolerant Multiprocessing
and Development Tools
Period of Work Covered: Dec 85 - Nov 88

Principal Investigator: Dr. Tse-yun Feng
Phone: (814) 865-7667

RADC Project Engineer: Mark D. Foresti
Phone: (315) 330-2925

Approved for public release; distribution unlimited.

This research was supported by the Strategic Defense Initiative Office of the Department of Defense and was monitored by Mark D. Foresti, RADC (COTC), Griffiss AFB NY 13441-5700 under Contract F30602-86-C-0005.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS N/A		
2a. SECURITY CLASSIFICATION AUTHORITY UNCLASSIFIED			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A			5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-89-268		
6a. NAME OF PERFORMING ORGANIZATION Pennsylvania State University		6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COTC)		
6c. ADDRESS (City, State, and ZIP Code) 121 Electrical Engineering East Building University Park PA 16802			7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Strategic Defense Initiative Office		8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-86-C-0005		
8c. ADDRESS (City, State, and ZIP Code) Office of the Secretary of Defense Wash DC 20301-7100			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO. 63223C	PROJECT NO. B413	TASK NO. 03
			WORK UNIT ACCESSION NO. 16		
11. TITLE (Include Security Classification) FAULT TOLERANT MULTIPROCESSING AND DEVELOPMENT OF TOOLS					
12. PERSONAL AUTHOR(S) Dr. T. Feng, Dr. C.R. Das, Dr. P.T. Hulina, Dr. A.R. Hurson, Dr. S. Pakzad, Dr. M.J. Thazhuthaueetil, Dr. C. Wu, Dr. D.L. Landis, Dr. W. Lin, Dr. J.J. Metzner					
13a. TYPE OF REPORT Finals		13b. TIME COVERED FROM Dec 85 TO Nov 88		14. DATE OF REPORT (Year, Month, Day) December 1989	
				15. PAGE COUNT 148	
16. SUPPLEMENTARY NOTATION N/A					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP			
12	07		Fault Tolerant Multiprocessing High Performance Development Tools		
19. ABSTRACT (Continue on reverse if necessary and identify by block number) Originally, Pennsylvania State University was to develop a conceptual framework for long life, fault tolerant, high performance space based computing system(s) for battle management. However, due to the reduced funding and redirection of the contract, Pennsylvania State University concentrated their research into the development of software tools. Pennsylvania State University has developed a hypercube performance model, an analytical simulation model for the butterfly network, a statistical code simulation, fault diagnosis for fault tolerant multistage networks, and algebraic coding techniques for distributed systems error control. This final report is an executive summary of all the work performed under this contract and summarizes twenty volumes of reports that were written by Pennsylvania State University.					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL Mark D. Foresti			22b. TELEPHONE (Include Area Code) (315) 330-2925		22c. OFFICE SYMBOL RADC (COTC)

DD Form 1473, JUN 86

Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

TABLE OF CONTENTS

	<u>Page</u>
1. REPORT SUMMARY	1
1.1. Objective.....	1
1.2. Report Organization.....	1
2. SOME RESULTS ON MULTISTAGE INTERCONNECTION NETWORKS.....	4
2.1. Introduction.....	4
2.2. Classification of Interconnection Networks.....	4
2.3. Shuffle-Exchange Networks.....	5
2.4. Generalized Baseline Networks.....	9
2.5. Matching Task Execution and Architecture.....	10
2.6. Routing Techniques.....	13
2.7. Conclusions.....	17
2.8. References.....	19
3. PERFORMANCE ANALYSIS OF FAULTY INTERCONNECTION NETWORKS	23
3.1 Introduction.....	23
3.2. Survey of Fault-Tolerance Techniques in Interconnection Networks Design.....	23
3.2.1. Techniques to achieve Fault-Tolerance.....	24
3.3. Problems in Fault-Tolerant Interconnection Network Design.....	24
3.4. A Fault-Tolerant Interconnection Network.....	25
3.4.1. A Network Architecture.....	25
3.4.2. A Routing Algorithm for Faulty Interconnection Networks.....	26
3.4.2.1. Problems of Conventional Routing Algorithm for Faulty Interconnection Networks.....	27
3.4.2.2. Proposed Routing Algorithm.....	27

3.4.2.3. Intermediate Terminal Selection Algorithm.....	28
3.4.3. A Fault Diagnosis Method for Interconnection Network.....	30
3.4.4. Performance Analysis of Faulty Interconnection Network.....	32
3.5. Summary and Directions.....	32
3.6. References.....	35
4. HIGH PERFORMANCE VLSI PROCESSOR ARCHITECTURES.....	36
4.1. Introduction.....	36
4.2. Dynamically Alterable Access Primitive (DALAP) Architecture.....	39
4.3. Advantages of the DALAP Architecture.....	44
4.4. Conclusions.....	44
4.5. References.....	47
5. VLSI BUILT-IN SELF TEST.....	48
5.1. Introduction.....	48
5.2. VLSI BIST Techniques.....	48
5.3. Application for BIST Techniques.....	49
6. A STATE OF THE ART RECONFIGURABLE DATABASE MACHINE.....	50
6.1. Introduction.....	50
6.1.1. Reconfigurability.....	50
6.1.2. Incomplete Data.....	51
6.1.3. Database Integrity and Consistency.....	52
6.1.4. Temporal Databases.....	54
6.2. Results.....	54
6.2.1. Hardware Reconfigurability.....	54
6.2.2. Handling Incomplete Data in ASLM.....	57
6.2.3. Enforcement of Integrity Constraints in ASLM.....	61

6.2.4. Handling Time in ASLM.....	62
6.3. References.....	63
7. DISTRIBUTED TRANSACTION PROCESSING	66
7.1. Introduction.....	66
7.1.1. Motivation for Distributed DBMS	66
7.1.2. Research Issues in DDBMS	67
7.1.3. Plan of the Report.....	67
7.2. The Distributed Transaction Processing Model.....	68
7.2.1. Transaction Processing.....	69
7.2.2. The 'Execute Construct.....	72
7.2.3. Proposed Model vs. the TM/DM Model.....	73
7.2.4. The Concurrency Controller	74
7.2.5. The Recovery Mechanism.....	76
7.3. Summary of the Qualitative Performance Analysis.....	77
7.4. The Implementation.....	82
7.5. Conclusions and Future Directions.....	83
7.5.1. Conclusions.....	83
7.5.2. Future Research Directions.....	83
7.6. References.....	85
8. SYSTEM LEVEL ERROR CONTROL IN FILE SYSTEMS	87
8.1. Introduction.....	87
8.2. Summary of Principal Results	87
8.3. Potential Applications.....	88
8.4. References.....	90
9. DEPENDABILITY TOOLS FOR BUTTERFLY AND HYPERCUBE COMPUTERS	91
9.1. Introduction.....	91
9.2. Evaluation of Existing Tools.....	92

9.2.1. CARE III	92
9.2.2. HARP	93
9.2.3. SHARPE	94
9.2.4. ARIES	95
9.3. Research Accomplishments	95
9.3.1. Butterfly Dependability	95
9.3.1.1. A Reliability Predictor for Butterfly	96
9.3.2. Hypercube Dependability	96
9.3.2.1. A Reliability Predictor for Hypercube	97
9.3.2.2. An Availability Predictor for Hypercube	98
9.4. Future Work	99
10. ALGORITHM MAPPING ON MULTISTAGE NETWORK PARALLEL COMPUTERS	101
10.1. Introduction	101
10.1.1. Objectives	101
10.1.2. Motivations	101
10.2. Parallel Conjugate Gradient Algorithm	103
10.3. Parallel QR Algorithm	104
10.4. Parallel Matrix Inversion Algorithm	106
10.5. Parallel Linear Programming Algorithm	108
10.6. Future Work	110
10.7. References	112
11. BUTTERFLY PERFORMANCE PREDICTOR	114
11.1. Introduction	114
11.2. Butterfly Performance Predictor Overview	115
11.3. Butterfly Parallel Processor	115
11.4. Code Simulator	119

11.5. Butterfly Simulator.....	122
11.6. Conclusion.....	123
11.7. References.....	126

Accession For	
NTIS CRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	



List of Figures

Figure 2.1	
A Partial Tree in the Space of Interconnection Networks.....	6
Figure 2.2	
A Shuffle-exchange network of size $N=16$	7
Figure 2.3	
A Switching Element.....	8
Figure 2.4	
A Recursive Process to Generate Baseline Topology.....	11
Figure 2.5	
System Topology.....	12
Figure 2.6	
An Example of the Interconnection Network.....	14
Figure 2.7	
(a) A Baseline network (B_N) (b) A Reconfigured Baseline Network	
(c) An Omega Network (B_{NoP}).....	15
Figure 4.1	
Intelligent Memory (MRU-Memory) Configuration.....	41
Figure 4.2	
CPU-MRU Configuration.....	41
Figure 4.3	
Dynamically Alterable Addressing Primitive Architecture	
(DALAP Architecture).....	43
Figure 6.1	
Average Execution Time of the Join Operation for Different	
Types and Number of Preprocessors.....	56
Figure 6.2	
Total Execution Time of the Join Operation for Different Type	
and Number of Preprocessors.....	58
Figure 6.3	
Average Execution Time vs. the Parallel Access Capability	
to the Auxiliary Storage.....	59
Figure 6.4	
Concurrency vs. Hardware Resources.....	60
Figure 7.1	
One Node of the Distributed Transaction Processing Model.....	70

Figure 7.2	
Hierarchy of Software Interfaces.....	73
Figure 7.3	
Logical Structure of the Proposed Model.....	75
Figure 7.4	
D_{trans} versus P_E	79
Figure 7.5	
N_{Doper} and D_{exec} for $X = N$ versus N	80
Figure 7.6	
D_{trans} versus C for $P_E = 1, 0.1, 0.2$ and 0.5 respectively.....	81
Figure 10.1	
The Configurable Multiprocessor System	102
Figure 10.2.1	
A General Communication Structure for Solving a Problem of Size n	105
Figure 10.5.1	
A Quadtree of 3 tree levels by the Basic Procedure.....	109
Figure 11.1	
Butterfly Performance Predictor.....	116
Figure 11.2	
Structure of Performance Predictor.....	117
Figure 11.3	
The Butterfly Parallel Processor.....	118
Figure 11.4	
A Butterfly Processor-Memory Node.....	120
Figure 11.5	
Sample Processor File.....	124

CHAPTER 1

REPORT SUMMARY

1.1. Objective

The objective of this research and development work was to design and specify a high performance, fault-tolerant spaceborne computer system architecture which provides reliable, secure execution of critical Battle Management algorithms. However, later in the program the effort was redirected to the software tool development.

During the course of this program, approximately 30 months, many volumes of reports, and analysis dealing with the many aspects of diverse systems requirements have been prepared and distribute to a limited number of personnel. It is our intent in this final report to summarize the various technical documents and to present one documentation set for the research/development effort.

1.2. Report Organization

This report is to provide a summary of each assigned task. The detailed technical work is reported in the subsequent volumes. They are listed as follows:

T. Feng and C. Wu, A Study of Some Multistage Interconnection Networks for Multiple-Processor Systems

S. Pakzad and Y. Chang, Performance Analysis of Faulty Interconnection Networks

P. Hulina and O. Morean, New Architectural Approaches for High Performance VLSI Processors

P. Hulina and O. Morean, Architectural Approaches for High Performance VLSI Processors

D. Landis, W. Check and D. Muha, VLSI Built-In Self-Test Support for the Fault Tolerant Battle Management Computing Environment

D. Landis and S. Raman, Classification and Evaluation of Built-In Self-Test Techniques for Fault-Tolerant VLSI Systems

A. Hurson, Performance Evaluation of an Associative Parallel Join Module

A. Hurson, Supporting Maybe Algebra in the Associative Search Language Machine (ASLM)

A. Hurson, L. Miller and S. Pakzad, Enforcement of Integrity Constraints in the Database Machine ASLM

A. Hurson, S. Pakzad and D. Shin, Extended ASLM* A Reconfigurable Database Machine

A. Elmagarmid and D. Mannai, A Distributed Transaction Processing System

J. Metzner and E. Kapturowski, Error Location Codes - Error Control Through Comparison of Replicated Files*

C. Das and J. Kim, An Analytical Model for Computing Hypercube Availability

W. Lin, C. Das and T. Sheu, A Parallel Conjugate Gradient Algorithm with Dynamic Communication Structures

W. Lin and T. Sheu, A Quadtree Communication Structure for the Butterfly Network

W. Lin and T. Sheu, A Parallel QR Algorithm with Adaptive Communication Structures

C. Das, W. Lin, M. Thazhuthaveetil and J. Kim, Reliability Evaluation of Hypercube Systems

W. Lin and C. Wang, Parallel Programming with Papa - A User's Guide

M. Thazhuthaveetil, A Butterfly Performance Predictor

CHAPTER 2

SOME RESULTS ON MULTISTAGE INTERCONNECTION NETWORKS

2.1. Introduction

Interconnection networks represent a consequent result of advances in computer technology and requirements for higher system performance. For the past three decades, computer systems evolved from 1960's batch processing models to 1970's time sharing models. Basically, the evolution was confined within von Neumann architectural model and hardware cost had been a limiting factor. Recent advances in VLSI technology have caused significant changes on the evolution. It is now economically feasible to construct a multiple-processor computer system by interconnecting a large number of off-the-shelf processor and memory modules. Furthermore, due to the higher performance requirements, the number of functional modules (homogeneous or heterogeneous) in the multiple-processor system would keep increasing as the domain of application enlarges. The challenge originates from at least two facts. First, the processing speed in the future can be significantly increased only by increasing the degree of the concurrent processing as speeds of computer devices reach a limit [1]. In addition, some classes of problems such as very large data base management, aerodynamic computation, etc., are beyond capabilities of current large computer systems. Thus, multiple-processor systems with as many as 10^5 processing elements have been proposed and constructed. However, conventional interconnection organizations such as time-shared buses, crossbar switches, and multiport memory schemes are not suitable for systems involving a large number of modules. Our earlier work [2] describes general design philosophy of the interconnection networks and then defines and investigates a class of multistage interconnection networks for the architectures of large scale multiple-processor systems. The significance of the work can be seen from not only the numerous citations in research journals and conferences, but also major inclusion in several books [3-5].

In this research we continue our previous work and achieve a number of useful technical results as summarized below.

2.2. Classification of Interconnection Networks

With numerous interconnection networks proposed and implemented it is important to have them divided into appropriate categories so that it is possible to perceive the advantages and limitations of a given interconnection network. We identify four dimensions of the space of interconnection networks: operation mode, control strategy, switching methodology, and network topology. The result of this classification is illustrated by Figure 2.1.

2.3. Shuffle-Exchange Networks

One of the major problems in designing multiple-processor systems for concurrent processing is to design a cost-effective interconnection network. The shuffle-exchange network has been shown to be a very good interconnection network in some particular applications [6-11]. A shuffle-exchange network of size $N=2^4$ is shown in Figure 2.2 in which the shuffle-exchange network is composed of two permutation connection patterns. One connection pattern is the perfect shuffle permutation and the other connection pattern is composed of $N/2$ 2×2 switching elements each of which can have straight connection as shown in Figure 2.3(a) and crossed connection as shown in Figure 2.3(b). The outputs of switching elements are connected to the inputs of the perfect shuffle connection pattern through intermediate registers.

The capability of realizing every arbitrary permutation with the shuffle-exchange network, or the universality of the shuffle-exchange network, has been investigated previously. On the basis of bitonic sorting, Stone [8] has shown an algorithm for realizing every arbitrary permutation in $O(\log_2 N)^2$ passes and Siegel [12] shows another in $2 (\log_2 N)^2$ passes where N is network size. With an additional mask facility, Lang and Stone [10,11] provides an upper bound of N shuffle-exchange passes for the universality. If we consider a pass of above schemes as a switching stage of $N/2$ 2×2 switching elements, compared to Benes binary network [13], these schemes involve many more passes than they should since the Benes binary network can realize every arbitrary permutation and needs only $2(\log_2 N)-1$ switching stages. Our study achieves a solution to realization of every arbitrary permutation in fewer shuffle-exchange passes. To realize an arbitrary permutation, we not only specify the number of shuffle-exchange passes needed, but also work out how to set the switching control for the realization. In addition to the realization scheme, we propose several universal

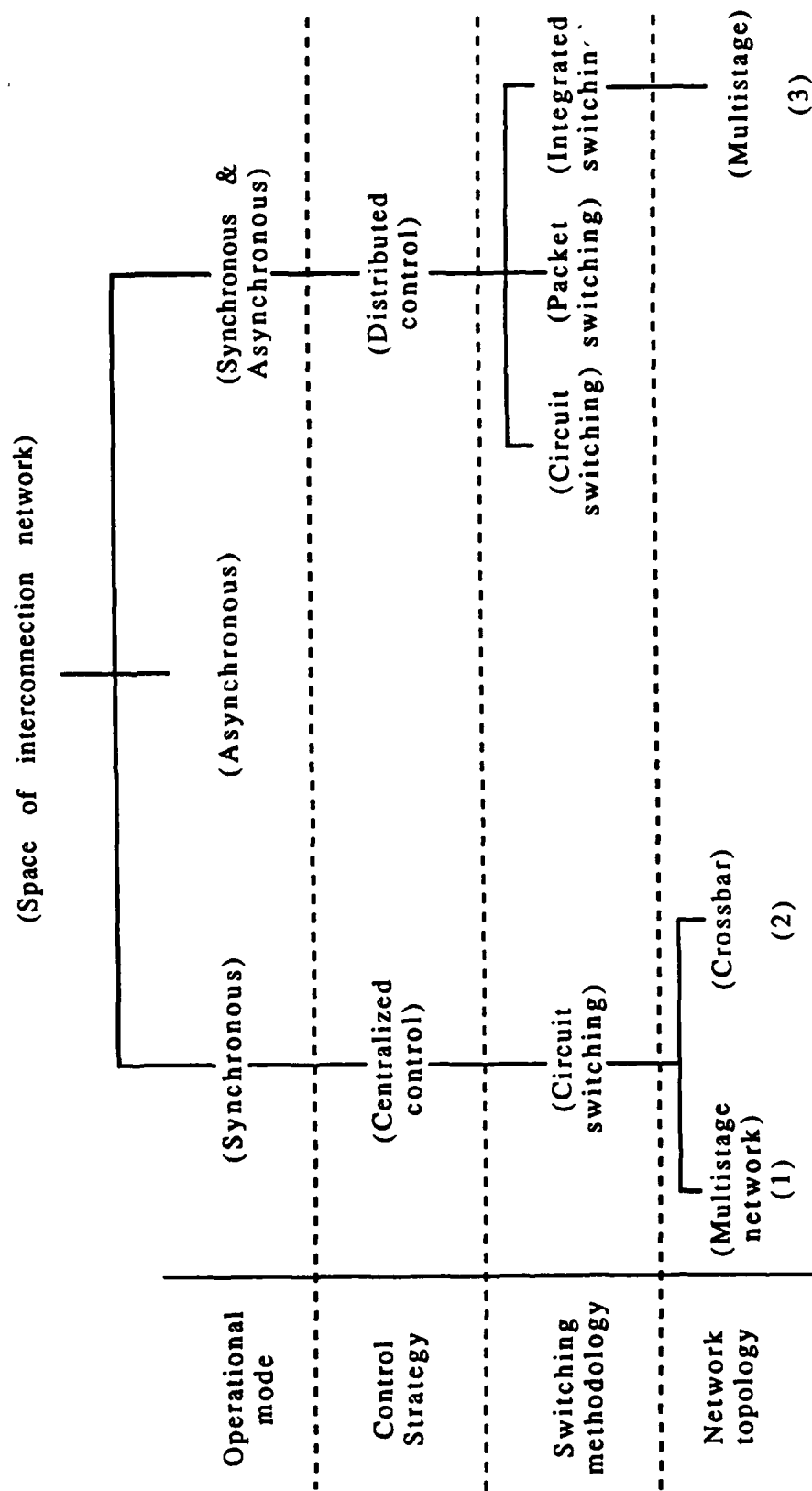


Figure 2.1. A Partial Tree in the Space of Interconnection Networks

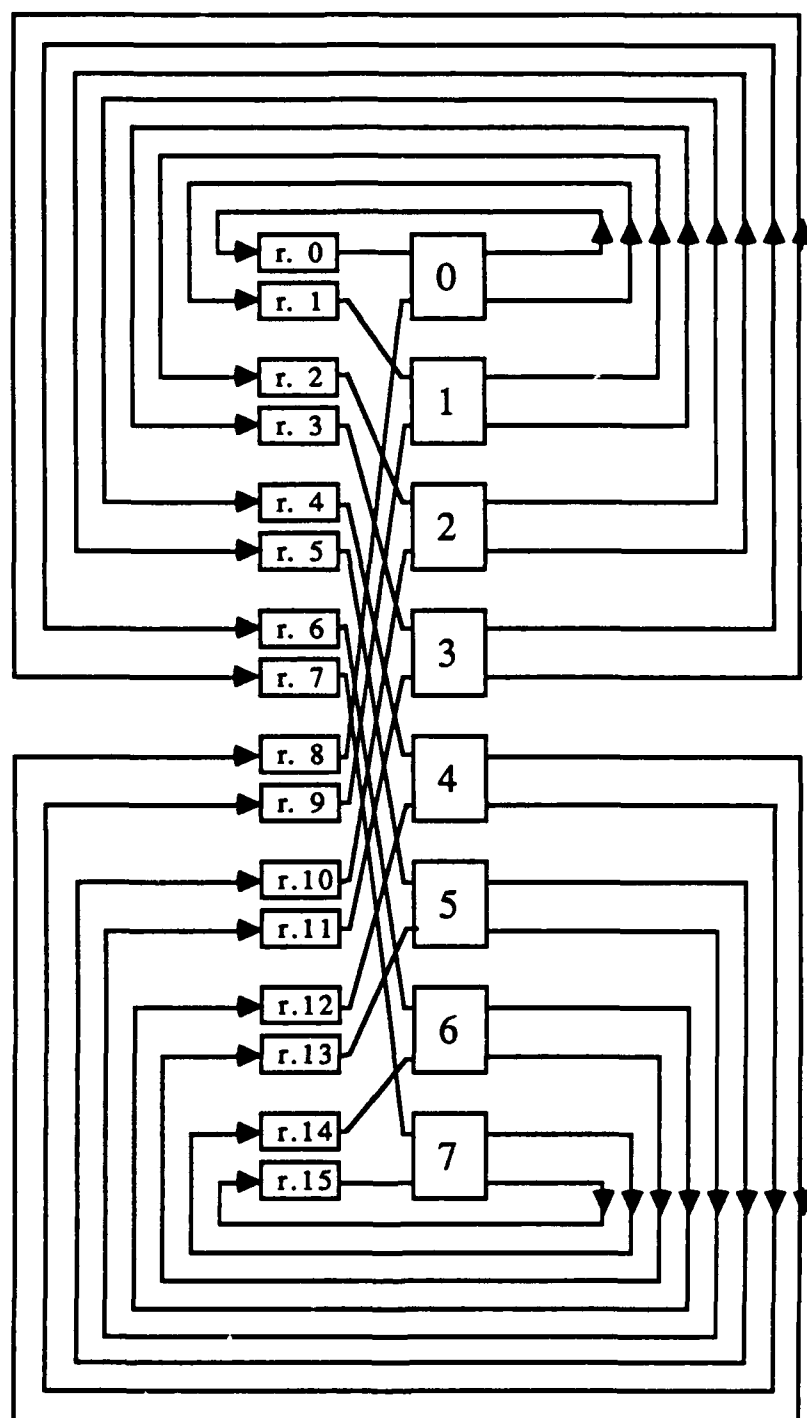
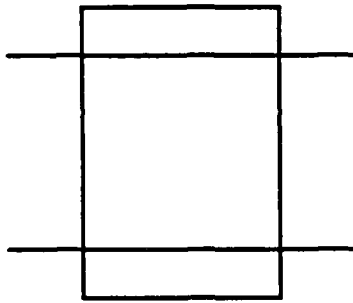
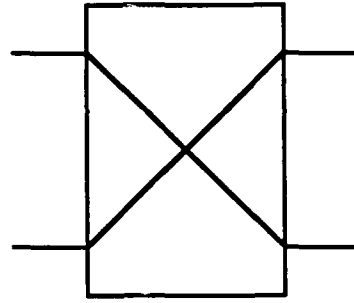


Figure 2.2. A Shuffle-exchange network of size $N=16$



(a)



(b)

Figure 2.3. A Switching Element

shuffle-exchange networks in terms of the permutation properties thus obtained.

It is worthwhile to mention here that Parker [14] uses Pease's result [15] to prove that omega network [9] can realize every arbitrary permutation in at most three passes (or $3 \log_2 N$ shuffle-exchange passes). Our work provides further contributions to the development of interconnection networks in several aspects. First, Parker uses the omega network as the basis since, as he claimed in his paper [14], the behavior of the omega network is simpler to analyze than that of bare shuffle-exchanges. Further improvement on Parker's work in terms of the omega network will reduce the three omega passes to the two omega passes, but not something between these two bounds (three and two omega passes). Our research enables us to use the single stage (bare) shuffle-exchange network as the basis and provide a bound of $3(\log_2 N) - 1$ shuffle-exchange passes which is the best bound of the state-of-the-art. Parker just shows a theoretical bound without specifying any control algorithm. Technically, given an arbitrary permutation, Parker can not provide a switching control setting for the realization of this permutation in three omega passes. We then provide a routing algorithm for our new scheme. With this routing algorithm, we can specifically develop a switching control setting to realize an arbitrarily given permutation in $3(\log_2 N) - 1$ shuffle-exchange passes. It should also be noted that our new scheme, along with its routing algorithm, specifies the inherent relationship between two well-known networks: the single stage shuffle-exchange network and the Benes binary network. Finally, our shuffle-exchange permutation properties can be used to demonstrate some optimized universal shuffle-exchange networks for better interconnection purposes.

2.4. Generalized Baseline Networks

Many multistage interconnection networks have been proposed in literature [9,16-25] to solve the intercommunication problem of multiple processor systems. These include a baseline network [21] which uses 2×2 switching elements as its composite module. Our research extends this version to a more flexible (and practical) one which allows the use of a more general $t \times r$ switching elements, $t, r \geq 1$, as the composite module which allows connecting more than one of idle outputs to an idle input, this broadcast capability can enhance the efficiency of the distributed processing. The topology of the

extended version is generated in a recursive way. Figure 2.4 shows the first iteration of the recursive process in which the first stage contains N/r versatile crossbar modules and the second stage contains t sub-blocks. A number of theorems are developed characterizing the extended baseline network to other networks.

2.5. Matching Task Execution and Architecture

The problem of reconfiguring a distributed computing system for enhancing the performance of executing computation tasks is important. Architecturally, a distributed computing system can be constructed by connecting off-the-shelf hardware components (processors and memories) with an interconnection network. To execute a computation task, system programmers have to develop a software system to partition the computation task into co-operating processes and to assign the pieces to processors [26]. However, the concepts and techniques for partitioning and assignment are considered very primitive [27]. It is even unclear what criteria should be used for general N -processor partitioning and assignment [28]. In addition, the communication need among the co-operating processes is now conceived at least to be as important as the computation need [29]. So far, how to match the execution of the task and the distributed processor communication architecture effectively is not well understood. A software technique is demonstrated to effectively match the task execution and the distributed architecture and consequently to enhance the performance without changing the hardware organization.

The topology of the distributed computing system under consideration is illustrated in Figure 2.5. In Figure 2.5, the P and M blocks represent the processor and the memory, respectively, and the central block represents the full-access interconnection network. The system topology shown in Figure 2.5 actually reflects only the logical structure of the system. A memory can be physically separated or associated with a processor. In either case, the memory and the processor can then have different logical names. The full-access interconnection network in the center plays a major role in the architecture. Its functionality not only influences the interprocessor communication directly, but also affects the way of processing tasks. Nowadays, people realize that conventional interconnection schemes such as time-shared bus, multiport memory and

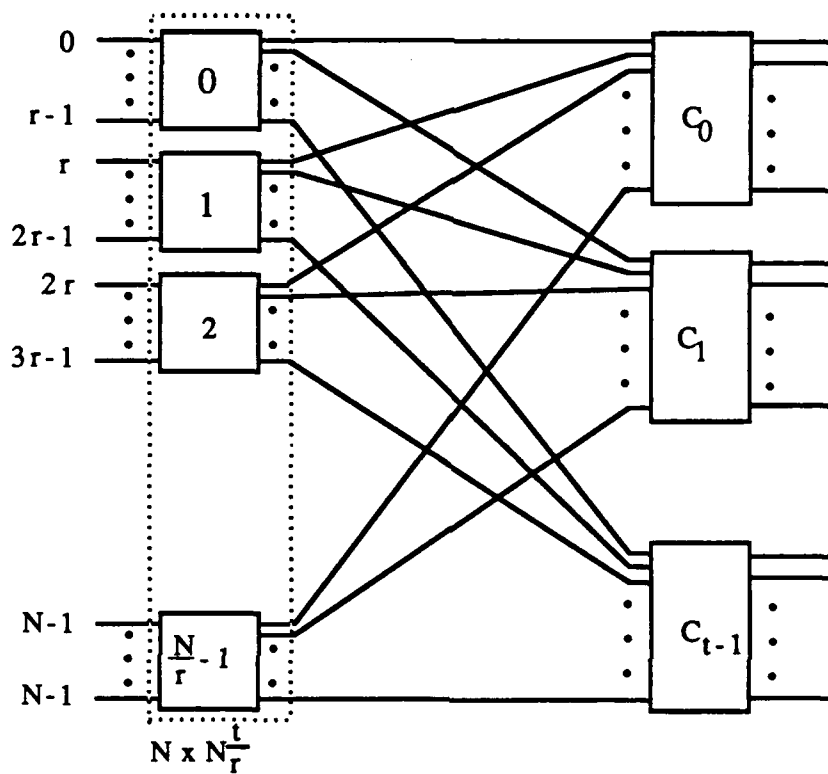


Figure 2.4. A Recursive Process to Generate Baseline Topology

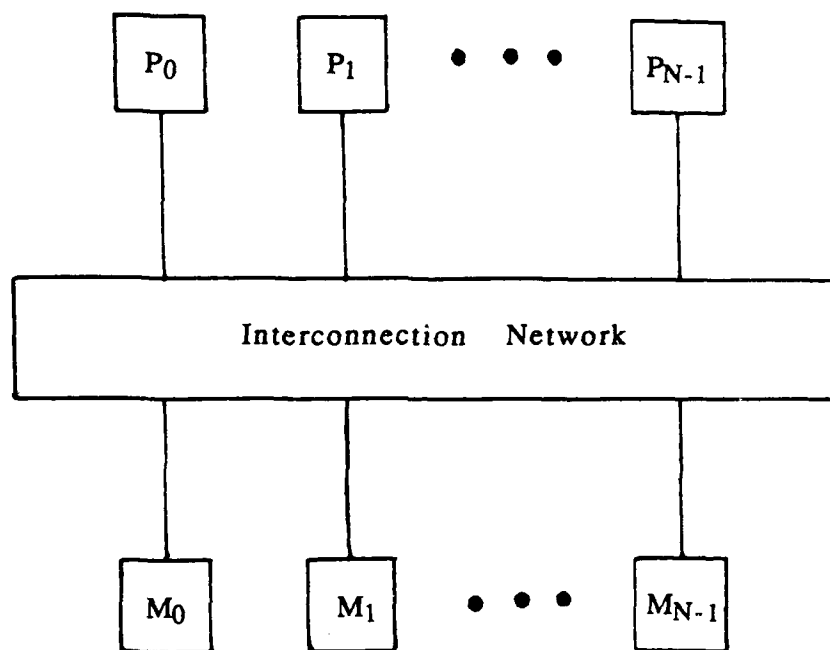


Figure 2.5. System Topology

crossbar switch become awkward in handling the intercommunication in large-scale systems [16, 30]. There is a class of multistage interconnection networks [21] which are potentially good for this intercommunication purpose. The possible performance enhancement in using this class of interconnection networks as the intercommunication vehicle is explored. Since the interconnection networks in the class are topologically equivalent, we can use the name of "baseline network" for discussion without losing generality. Figure 2.6 shows an example of the baseline network for $N = 8$ where N is the number of input (or output) ports of the network. An input port is connected to an active module such as the processor and an output port is connected to a passive module such as the memory. Figure 2.6 also shows an identify feedback connection from the intermediate output register to the input port. The feedback connection is for a universal two-pass structure [31] which can realize every arbitrary permutation in two passes.

We then address the communication complexity of our interconnection network and a technique to reduce the communication complexity. As an example of applying the theory developed in our work, we show an approach for implementing parallel algorithms with the network configurations, and we make a unified discussion on some frequently mentioned network configurations in terms of some useful conflict-free permutations (Figure 2.7). We also show a recursive external control scheme for those conflict-free permutations.

2.6. Routing Techniques

A number of routing techniques for rearrangeable interconnection networks useful for parallel/distributed processing systems are investigated. In a parallel/distributed processing system, an interconnection network is favorably used to facilitate a mixed-mode operation of SIMD, multiple SIMD and MIMD. In the SIMD environment, parallel algorithmic processes usually need simultaneous one-to-one connections (or permutation) between processors. The permutation connections are established synchronously. Many interconnection networks have been proposed for permutation connections [21]. However, these proposed interconnection networks can not realize every permutation in a single pass. In the multiple SIMD environment, the interconnection network must be partitioned so that each partition can serve the need of the SIMD permutations.

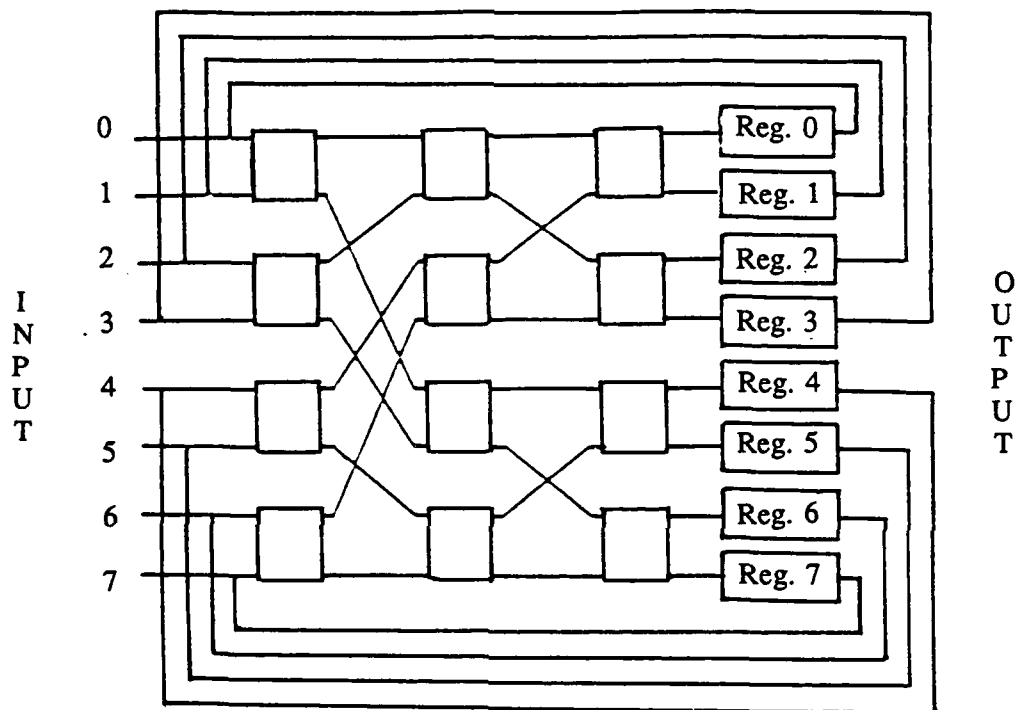
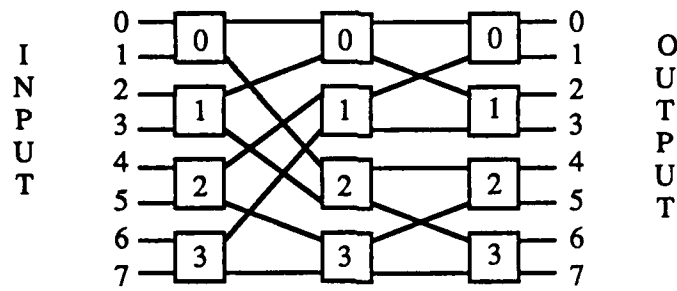
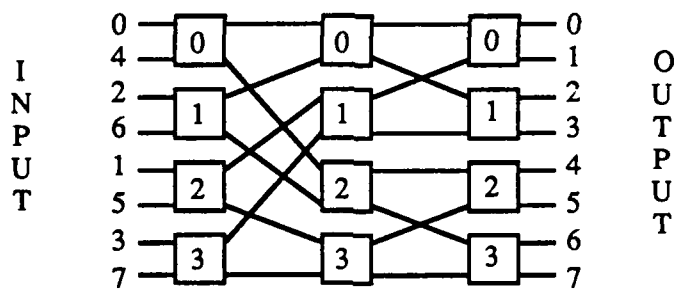


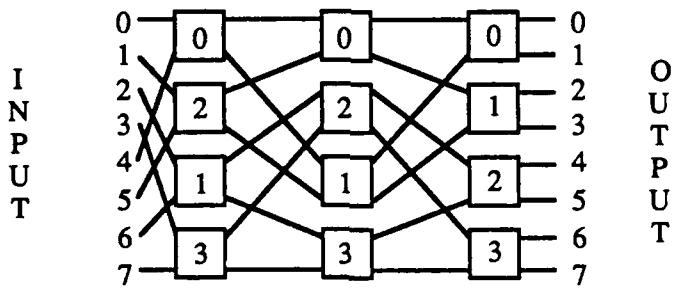
Figure 2.6. An Example of the Interconnection Network



(a)



(b)



(c)

Figure 2.7. (a) A Baseline network (B_N) (b) A Reconfigured Baseline Network (c) An Omega Network (B_{Nop})

Siegel [32] and associates have done useful work on the partition of these networks mentioned in [21]. Nevertheless, the size of each partition has to be equal to a power of two. On the other hand, in the MIMD environment, connection requests are likely to occur dynamically and asynchronously. In regard to this stochastic property, we like to have an interconnection network which can provide a connection path for every connection request no matter what the current status of the interconnection network. A crossbar switch can certainly fulfill these prescribed needs: one-pass realization of arbitrary permutation, multiple SIMD machines of an arbitrary size, and the non-blocking requirement for asynchronous requests. However, the cost of a crossbar switch grows as N^2 , where N is the number of input (or output) lines. Rearrangeable networks [33] can also fulfill these needs for the mixed-mode operation and yet its cost grows as $N \log N$. But, one main issue concerning the use of the rearrangeable networks is the limitation induced by backtracking routing algorithms.

Many routing algorithms have been developed for the control of rearrangeable networks. Opferman and Tsao-Wu [34] developed a looping algorithm to control a class of rearrangeable networks with 2^n input (or output) lines (where n is an integer). Anderson [35] extended the looping algorithm for the network with $2^t \cdot n_t$ input lines (where t and n_t are both integers). However, Anderson's extension work requires a working memory of size $(2^t \cdot n_t) \times (2n_t)$. To control rearrangeable networks with an arbitrary size, Neiman [36], Ramanujam [37], and Hall [38] provided different algorithms which are all backtracking in nature. These routing algorithms need large working memory and require long processing time for backtracking. In order to fulfill constraint of computer systems, better routing techniques should be solicited. In addition to the lack of a better routing technique, there appears to be no algorithm which takes the needs of asynchronous routing into consideration.

In our report, we develop both synchronous and asynchronous routing techniques for rearrangeable interconnection networks. We first present a graph model to describe general routing for the rearrangeable interconnection networks. Efficient implementations of the algorithms are illustrated and discussed. We then use the same data structure derived from the graph model to study strategies for asynchronous operations, and simulation models and the

results of analysis are discussed. Finally, the graph routing is modified to adapt full processor communication.

2.7. Conclusions

Four dimensions (operation mode, control strategy, switching methodology, and network topology) are identified which constitute the space of interconnection networks. The cross product of above four dimensions then represents the space. Along this taxonomy, contemporary research works on interconnection networks can easily be categorized.

For our work on shuffle-exchange networks it is concluded that every arbitrary permutation can be realized in at most $3(\log_2 N) - 1$ shuffle-exchange passes where N is the network size. A routing algorithm for this new scheme is also provided. Three optimized networks in various forms to further reduce the shuffle-exchange passes are proposed. In view of previous works on the shuffle-exchange network, the present work specifically provides several contributions. First, the new upper bound of $3(\log_2 N) - 1$ shuffle-exchange passes for the universality is better than other developments. Second, the routing algorithm is good for realizing every arbitrary permutation in $3(\log_2 N) - 1$ passes while other routing algorithms available most recently are developed for the realization of $(\log_2 N)^2$ shuffle-exchange passes. Third, this work specifies the inherent relationship between the shuffle-exchange network and the Benes binary network. Finally, three expanded universal networks which can provide better interconnection schemes are demonstrated.

On the generalized baseline network, its relationships to other networks proposed in other contemporary research efforts are studied. It is concluded that the results obtained under different network names are mutually applicable. A software technique is then demonstrated which can reduce the time delay in realizing communication needs of a computation task without changing the hardware organization. The software technique involves reconfiguring the interconnection network so that the routing delay for the required communication can be minimized. The reconfiguration mechanism can result in a flexible architecture so that system programmers can have options to better partition the computation task and assign pieces to processors and memories.

Finally, a graph routing algorithm for the rearrangeable interconnection network is presented. The algorithm can be implemented on a data structure which uses small amounts of memory and is good for a network of arbitrary size in contrast to other algorithms. Both synchronous and asynchronous routings can be implemented on the same data structure. The algorithm provides good data structure for implementation using associative memories, heuristical method to degrade backtracking, and simulation of various asynchronous operations.

Several future researches are recommended as follows:

- (1) Reliability. This issue receives inadequate support although it is one of the most important aspects of interconnection networks.
- (2) VLSI implementation. The necessity of the interconnection network comes from advances of VLSI technology. Its implementation is a logical step toward an efficient system realization.
- (3) Establishment of evaluation metrics. The cost function must be defined in order to have an objective comparison among various schemes.
- (4) Stochastic simulation. There is relatively small amounts of research activity on the stochastic behavior of interconnection networks. Performance evaluation of various networks design decisions through the stochastic simulation can provide useful information for multiprocessing architecture.

2.8. References

- [1] T. Feng, "An Overview of Parallel Processors and Processing," ACM Computing Surveys (March 1977), pp. 1-2.
- [2] T. Feng and C. Wu, Interconnection Networks in Multiple-Processor Networks, RADC-TR-79-304, 230 pp., 1979.
- [3] H.J. Siegel, Interconnection Networks for Large-Scale Parallel Processing, Lexington Books (1985), 260 pp.
- [4] K. Hwang and F.A. Briggs, Computer Architecture and Parallel Processing, McGraw-Hill (1984), 846 pp.
- [5] T. Feng and C.L. Wu, Interconnection Networks for Parallel and Distributed Processing, Computer Society Press (1984), 647 pp.
- [6] S.W. Golomb, "Permutations by cutting and shuffling," SIAM REV., Vol. 3, Oct. 1961, pp. 293-297.
- [7] M.C. Pease, "An adaptation of the fast Fourier transform for parallel processing," J. Association Comput. Machinery, Vol. 15, April, 1968, pp. 252-264.
- [8] H.S. Stone, "Parallel processing with the perfect shuffle," IEEE Trans. Comput., Vol. C-20, No. 2, Feb. 1971, pp. 153-161.
- [9] D.H. Lawrie, "Access and alignment of data in an array processor," IEEE Trans. Comput., Vol. C-24, Dec. 1975, pp. 1145-1155.
- [10] T. Lang and H.S. Stone, "A shuffle-exchange network with simplified control," IEEE Trans. Comput., Vol. C-25, No. 1, Jan. 1976, pp. 55-65.

- [11] T. Lang, "Interconnections between processors and memory modules using the shuffle-exchange network," IEEE Trans. Comput., Vol. C-25, No. 5, May 1976, pp. 496-503.
- [12] H.J. Siegel, "The universality of various types of SIMD machine interconnection networks," Fourth Annual Symp. Computer Architecture, March 1977, pp. 70-79.
- [13] V. Benes, Mathematical theory of connecting networks, New York: Academic Press, 1965.
- [14] D.S. Parker, "Notes on shuffle/exchange-type switching networks," IEEE Trans. Comput., Vol. C-29, No. 3, March 1980, pp. 213-222.
- [15] M.C. Pease, "The indirect binary n-cube microprocessor array," IEEE Trans. Comput., Vol. C-26, No. 5, May 1977, pp. 458-473.
- [16] H.C. Pease, III, "The indirect binary n-cube microprocessor array," IEEE Trans. Comput., C-26, May 1977, pp. 458-73.
- [17] T. Feng, "Data manipulating functions in parallel processors and their implementations," IEEE Trans. Comput., C-23, No. 3, March 1974, pp. 309-18.
- [18] R. Goke and G.J. Lipovski, "Banyan networks for partitioning multiprocessor systems," Proc. 1st Conference on Comput. Arch., Dec. 1973, pp. 21-28.
- [19] G.J. Lipovski and A. Tripathi, "A reconfigurable varistructure array processor," Proc. of the 1977 International Conference on Parallel Processing, pp. 165-74.
- [20] K.E. Batcher, "The flip network in SARAN," Proc. of the 1976 International Conference on Parallel Processing, pp. 65-71.

- [21] C. Wu and T. Feng, "On a class of multistage interconnection network," IEEE Trans. Comput., C-29, pp. 694-702, Aug. 1980; also see Proc. of the 1978 International Conference on Parallel Processing, pp. 197-205.
- [22] F. Briggs, K.S. Fu, K. Hwang and J. Patel, "PM4-a reconfiguration multimicro-processor system for pattern recognition and image processing," AFIPS Conference Proc., 1979, pp. 255-65.
- [23] J.H. Patel, "Processor-memory interconnections for multiprocessors," Proc. 6th Annual Symposium on Computer Architecture, pp. 168-77.
- [24] T. Feng, C. Wu, and D. Agrawal, "A microprocessor-controlled asynchronous circuit switching network," Proc. 6th Annual Internat'l. Symp. on Comput. Arch., April 1979, pp. 202-15.
- [25] H.J. Siegel, L.J. Siegel, R.J. McMillen, P.T. Mueller, Jr., and S.D. Smith, "An SIMD/MIMD multimicroprocessor system for image processing and pattern recognition," 1979 IEEE Comp. Soc. Conf. Pattern Recog. and Image Processing, pp. 214-24.
- [26] E.D. Jensen and W.E. Boebert, "Partitioning and assignment of distributed processing software," Proc. COMPCON Fall 1976, pp. 348-52.
- [27] E.D. Jensen, "The honeywell experimental distributed processor--an overview," Computer, Jan. 1978, pp. 28-38.
- [28] R.H. Eckhouse, Jr. and J.A. Stankovic, "Issues in distributing processing--an overview of two workshops," Computer, Jan. 1978, pp. 22-26.
- [29] T. Agerwala and B. Lint, "Communication issues in parallel algorithms and systems," Proc. COMPSAC '78, pp. 583-88.

- [30] H. Sullivan, et al., "A large scale, homogeneous, fully distributed parallel machine," Proc. of the 4th Annual Symposium on Computer Architecture, pp. 105-24.
- [31] C. Wu and T. Feng, "The reverse-exchange interconnection network," IEEE Trans. Comput., Vol. C-29, No. 9, pp. 801-11, Sept. 1980; see also Proc. of 1979 International Conf. on Parallel Processing, pp. 160-74.
- [32] H.J. Siegel and S.D. Smith, "An interconnection network for multimicro-processor emulator systems," Proc. 1st International Conference on Distributed Computing Systems, 1979, pp. 772-782.
- [33] V.E. Benes, "Optimal rearrangeable multistage connection networks," B.S.T.J., 43, part 2 July 1964, pp. 1641-56.
- [34] D.C. Opferman and N.T. Tsao-Wu, "On a class of rearrangeable switching networks," B.S.T.J., 50, May-June 1971, pp. 1579-1618.
- [35] S. Andersen, "The looping algorithm extended to base 2^t rearrangeable switching networks," IEEE Trans. on Commun., Vol. COM-25, No. 10, October 1977, pp. 1057-63.
- [36] N.T. Tsao-Wu, "On Neiman's algorithm for the control of rearrangeable switching networks," IEEE Trans. on Commun., Vol. COM-22, No. 6, June 1974, pp. 737-42.
- [37] H.R. Ramanujam, "Decomposition of permutation networks," IEEE Trans. on Comput., July 1973, pp. 639-43.
- [38] M. Hall, Jr., "An algorithm for distinct representatives," Mathematical Notes, Vol. 63, 1956, pp. 716-17.

CHAPTER 3

PERFORMANCE ANALYSIS OF FAULTY INTERCONNECTION NETWORKS

3.1. Introduction

The interconnection network [4,5] is a major component of parallel computer systems. This report focuses on the fault-tolerant interconnection networks, the highly reliable connection networks between computer modules, and investigates the design and analysis of fault-tolerant interconnection networks.

We can define a fault-tolerant interconnection network as follows: A fault-tolerant interconnection network is a network that can tolerate faults of some degrees and still provide reliable connections between input-output terminal pairs.

Even though the fault-tolerant characteristics are important for the interconnection networks, not enough attention has been given to the networks in general. The design of fault-tolerant interconnection network involves the finding of effective methods that will allow the network to operate properly in the presence of some faults.

The report is organized as follows: In section 3.2, we will introduce several techniques to achieve fault-tolerant interconnection network design. In section 3.3, we will describe major problems which we have identified in fault-tolerant interconnection area as the results of our literature survey. In section 3.4, we will propose a comprehensive interconnection network design which includes network architecture, fault diagnosis method, control strategy, and performance evaluation. In section 3.5, we will summarize overall result of our research and propose directions in future works.

3.2. Survey of Fault-Tolerance Techniques in Interconnection Networks Design

To make interconnection network fault-tolerant, several techniques have been proposed. This chapter focuses on several major techniques to achieve fault-tolerance.

3.2.1. Techniques to Achieve Fault-Tolerance

In the past, several multistage interconnection network (MIN) with fault-tolerant characteristics have been proposed. The examples of such designs are the extra stage cube network, the chained multistage interconnection network and the interconnection network using error-correcting codes, etc. Here we can identify four major ways to achieve fault-tolerant multistage interconnection networks:

First, utilizing multiple alternative paths between an input-output pair of the MIN. We can either modify existing network topology [1,9] to obtain redundant paths or derive new topology that can be reconfigured to avoid faults with little degradation of performance.

Second, using multiple-pass routing scheme. We can implement a feedback loop between input-output terminal [8] and route data several times through feedback loop.

Third, adapting the fault-tolerant switching elements and control units to construct fault-tolerant multistage interconnection network.

Fourth, using self-checking and correcting code for the data transmitted through the network. We can use error correcting codes that can correct the errors due to failure of switching elements and connecting links.

With these techniques, it is easy to understand that the fault-tolerance characteristics can not be separated from interconnection network characteristics, since once an interconnection network is constructed, the fault-tolerant characteristics can not be changed greatly.

3.3. Problems in Fault-Tolerant Interconnection Networks Design

The background survey of this report has attempted to identify the problem, and develop the basis for fault-tolerant interconnection networks. As a consequence of this survey, we discovered the problems and untouched issues in previous researches. Those issues are summarized below.

- Analyzed with emphasis only on theoretical characteristics such as fault-tolerant parameter and network isolation due to fault. Therefore

the problems such as how the network can be controlled with faulty elements are not solved.

- Designed networks have either extra stages or extra links between switching elements. As a result they create resource under utilization problem for the valuable resources like switching elements and links, since these extra hardware will sit idle during normal fault-free operation time.
- Introduced irregular elements for the network such as limited 3×3 switching element and special types of data manipulating device, which might create implementation problems.
- Most of the design can only handle single fault problem does not provide general multiple faults coverage.
- Control algorithms of proposed networks are not regular and frequently requires time consuming operations like alternative path inspection. These inefficient control schemes will degrade the network performance greatly.

Therefore we are now proposing a comprehensive design of a fault-tolerant interconnection network which includes network architecture, fault diagnosis, and control strategy. Some of the merit of our design are:

- Since no extra switching hardware are involved, the resource utilization will be very high. Also normal fault-free operation of our network does not create any overhead in hardware and software control.
- New proposed routing algorithm can handle both fault-free network and network with faulty elements with regular fashion.
- Our design can handle general multiple faults in the network, where other designs only can handle a single fault and very limited multiple faults problem.

3.4. A Fault-Tolerant Interconnection Network

3.4.1. A Network Architecture

The network architecture described in this chapter is based on the baseline networks. Baseline networks are a class of unique-path networks, which provide exactly one path for every pair of the input and output terminals (full access network). Since the unique-path networks are a minimal network which still provides the full access property, a fault in the network will prevent some connections from being established.

By adding a feedback loop between output terminals and input terminals, a unique-path network can tolerate a number of faults. This feedback loop already exists in many multicomputer systems. Even if it does not exist, the extra hardware needed to the system is negligible compare to other designs which usually added either extra stages or links. The feedback loop interconnection network architecture has been proposed and investigated in several previous works [8]. But most of the previous works only deals with characteristics of the network and the problem of controlling the network such as routing data in faulty network is yet to be addressed.

The fault model [6,7] discussed in this paper is the switching element stuck-at-fault type. A switching element can be stuck-at either Through or Cross mode. Therefore, a faulty switching element has lost its 'switching' capability, and can only pass messages in a fixed pattern.

3.4.2 . A Routing Algorithm For Faulty Interconnection Network

In this section, we will introduce problems of conventional destination tag routing algorithm when it is applied to control faulty interconnection networks, and then propose a new routing scheme which can control faulty interconnection network as well as fault-free networks.

Since the possible variations of multiple faults pattern is almost infinite, finding efficient take-care-everything control scheme is almost impossible. Therefore we approach the problem with two step strategy. First, we employ a heuristic approach to control the network by observing the network activity. Second, we employ a deterministic approach to insure every input-output connection requirements are established. In the first step, we control the network according to the output observation and detected faults by several heuristic rules. Since the first step may not take care of all the cases, the second step is designed for the unsolved cases. In the second step, an intermediate

terminal, through which the source/destination connection can be established, will be found. Then the messages will be routed through the intermediate terminals. The cost of the deterministic approach is higher compare to the heuristic approach, since the former approach requires more processing time. But from the simulation of actual network operation with faulty elements, we found out only small portion of the test cases requires the second step and most of the time the first step is sufficient.

3.4.2.1. Problems of Conventional Routing Algorithm For Faulty Interconnection Networks

Two problems are identified when the original destination tag algorithm is applied to faulty networks: The fault propagation problem and the infinite loop problem.

The fault propagation problem is so named because a fault in one switching element may appear to propagate through other switching elements in the network. If a message encountering a faulty switching element is misrouted, then conflicts may arise along its misrouted path. The misrouted, then conflicts may arise along its misrouted path. The misrouted message may collide with other messages trying to use switching elements along this path. Therefore, a greater number of problems may arise than we would expect from a single faulty switching element.

The infinite loop problem arises using the original destination tag algorithm when a group of messages cannot reach their destinations. In an infinite loop situation, the terminals which the unsuccessful messages can reach form a group. However, the intended destinations lie outside this group, and additional re-routing can not help. Consequently, these messages can never be successfully delivered.

3.4.2.2. Proposed Routing Algorithm

As previously stated, the first step of our routing algorithm for faulty networks, is based on the destination tag algorithm for fault-free networks. We have made three modifications to directly address the fault propagation and infinite loop problems.

First, whenever two messages meet and create a conflict at a faulty switching element, they will exchange their destination tags. As a result, although these two messages will not reach their correct destinations, they will not stray into the paths of any additional messages. Therefore, only the original two messages are affected by the fault, and the fault propagation problem is eliminated.

Second, we have found that the infinite loop problem mainly occurs when a misrouted message reaches an incorrect destination which belongs to a switching element, in the same row as the source. To avoid an infinite loop, our algorithm re-routes this message to another part of the network for the next pass. Re-routing is accomplished by complementing the two leftmost bits of a message's destination tag. The logic behind this procedure is to send the message away from the faulty part of the network which appears to be the cause of the infinite loop problem. Since the two leftmost bits partition the network into quarters, complementing these bits will route the message to a different quarter of the network. For a single fault network, then, the message is re-routed to a fault-free portion of the network, and delivery to the correct destination is assured with one more pass. For a multiple fault network, the message is re-routed to a portion of the network with potentially fewer faults. Therefore, the probability of reaching the correct destination in the next pass is quite high.

However, occasionally, the proposed procedure does not eliminate the infinite loop problem for multiple fault networks. Hence, an algorithm was designed to handle these rare occasions. The possibility of an infinite loop can be detected when a pass through the network does not reduce the number of undelivered messages. When such a condition is detected, our algorithm finds an appropriate intermediate terminal for each undelivered message. By re-routing the message through this terminal, the destination can now be reached. The algorithm for selecting the appropriate intermediate terminal is discussed in the next section.

3.4.2.3. Intermediate Terminal Selection Algorithm

In this section we will describe the second step of our proposed routing algorithm.

Whenever a pass through the network does not reduce the number of undelivered messages there is a potential infinite loop problem. When such a condition exists, we choose a new route for each undelivered message by selecting an intermediate terminal through which it can be successfully delivered.

For a given fault status (location and type of faulty switching elements) and a given/source/destination pair this selection process has three phases:

1. Determination of all possible reachable destination terminals from the given source terminal.
2. Determination of all possible source terminals for the given destination terminal.
3. Intersection of these two sets to determine the set of appropriate intermediate terminals.

Since we may have several sets of source/destination pair, we need one more phase when there are more than one source/destination pairs which needs the intermediate terminal selection scheme. The fourth phase is:

4. Pick a compatible intermediate terminal for each source/destination pair which will not create conflicts with any other messages.

We call this overall process the Intermediate Terminal Selection Algorithm. The first phase determines the set of destination terminals which are reachable from the source terminal. The general approach is to build a binary (trace) tree with the source terminal as the root node and the reachable destinations as the leaf nodes. We call this phase the Forward Trace Algorithm. The trace algorithm is based on the connection topology of the network. In the case of fault-free baseline network, the subroutine generate-switch-set-for-next-stage constructs a binary trace tree for the given source terminal. For faulty network, the trace algorithm splits the trace tree after the detection of each faulty switching element. The details for the forward trace will be in the final report.

The second phase of the trace algorithm determines the set of source terminals which can reach the required destination terminal. The approach here is to build a binary (trace) tree with the destination terminal as the root node and possible source terminals as the leaf nodes. This algorithm, called the Backward

Trace Algorithm. is simply the reverse baseline network. Therefore, the subroutines for generating the switch descriptor set for the next stage and splitting the faulty node should be adjusted for the new topology. For space considerations, then, this algorithm will not be described here.

The third phase of the trace algorithm determines the set of intermediate terminals through which a message can pass if its original path is blocked by a faulty switching element. This set of terminals is simply the intersection of the terminal descriptor sets from the first two phases. And the possible intermediate terminal is any one of the intermediate terminal set. When there is only one source/destination pair, the intermediate terminal can be indeed any terminal in the set. But for the case of multiple source/destination terminals, we need to pick a terminal for each pair which does not create any conflict with the other messages.

When there are several source/destination terminals which need intermediate terminal, we will need one more step to find the compatible intermediate terminals among the sets of possible intermediate terminals. We have developed heuristic rules which will reduce the change of the conflict between the messages. Those rules are:

First, pick a terminal from the smallest terminal descriptor in the backward trace tree which is also in intermediate terminal set. This rule force the message pass through the faulty switching elements. Since the message pass through more clearly defined passage by the faulty switching elements, there is less possibility of creating a new conflict with other messages. Second, for remaining sets of source/destination pairs, pick a terminal descriptor far from already picked terminal descriptors. This second rule is for separating the messages from each other. Therefore there will be less chance of a conflict. Third, repeat second rule, until all the intermediate terminals for each source/destination pair are found.

3.4.3 . A Fault Diagnosis Method For Interconnection Network

This section describes a fault diagnosis method which detects and located the faulty elements in the network. The network routing algorithm described in the previous section will utilize the fault information obtained by the fault diagnosis scheme explained below. Our fault diagnosis method is a combination of

hardware and software diagnosis method. We will explain the hardware diagnosis part first, and will describe the software part later.

A simple technique has been developed based on fault model and routing scheme. Our fault model are the switching element stuck-at fault, where the switching elements can be stuck at either through setting or cross setting. So the faulty switching element will be set at one of the setting and can not be changed by the control signal. Our scheme is based on the destination tag of the messages which pass through the 2×2 switching element in each stage.

A fault detection logic circuit, which is described in detail in the final report, in each switching element will detect the fault and identify the type of the fault, then generate fault signal and fault type signal. These fault signals for each switching element will be fed into the inputs of fault controller of the network partition which is composed of a simple logic circuit such as a multi-input OR gate and sequential logic. The function of network partition is defined as a group of switching elements which are closely related in the network topology and control method. Since integrated circuit (IC) chip building technology limit the number of switching element can be put inside a single chip, it is natural to make a network partition which can be fit in a single chip. By having one network partition fault controller in each chip, we could save the valuable resource, the outside connection pins which is one of the major problems of current IC technology.

For the software diagnosis part of our fault diagnosis method, one of our previously proposed method [2] can be used. The essence of these types of fault diagnosis method is applying the input test vector which will create distinguishable output patterns when some faulty elements are present. Since we assumed two valid states for the switching element, we need at least two steps to test the network. The major improvement over the previously proposed fault diagnosis method is the activation time of the fault diagnosis routine. A major problem of pure software fault diagnosis techniques [1,3], either off-line or on-line technique, is the activation time of fault diagnosis. It is almost impossible to find a optimal timing for each fault diagnosis. To solve this problem, we propose to combine the software and the hardware diagnosis techniques. The software routine will be activated only if the master controller informs a new fault has been detected.

In summary, we feel that our fault diagnosis method is quite an improvement over the previously developed methods. Also it seems our design is flexible enough with current IC technology.

3.4.4. Performance Analysis of Faulty Interconnection Network

The performance measure we have used here is based on the communication delay. The communication delay, which represents the total delay time for realizing every source/destination connection (permutation) requirements, is used as the measure of the performance of the networks. Since our network is multistage interconnection network with feedback loop between destination and source terminals, the number of passes requires to realize a source/destination permutation will be used as our performance measure.

Because the variations of the multiple fault patterns are almost unlimited, we have employed a simulation technique to collect statistical data for our analysis. We are simulating the baseline networks with four different sizes, which are 16 X 16, 32 X 32, 64 X 64, and 128 X 128 baseline networks. The number of fault we are testing are up to (but not included) the minimal critical fault. For example, for 64 X 64 baseline networks, we are testing faults number starting from one fault up to five faults. For each test case (a chosen network size and number of faults) we are testing several thousand different test data to gather the statistical data. Our partial results of the simulation have been very encouraging. Our design yields graceful performance degradation in the presence of network faults. For every network size we tested, on average, less than two passes are required for a single fault network. Even with the number of fault is increased up to the minimal critical fault, the number of passes required is about 3.5 to realized a source/destination permutations. For example, the 128 X 128 baseline networks with six faulty switching elements only required average 3.5 passes to establish all the source/destination connections. The detailed results of our simulation will be shown in the final report.

3.5. Summary and Directions

In this chapter, we will summarize this report and propose the directions for the future works.

First, we have described major problems which we identified in fault-tolerant interconnection are as the results of our literature survey. Second, we have proposed a comprehensive interconnection network design which includes a network architecture, fault diagnosis method and control scheme of the network and our design is readily applicable for many real world applications. In our design, the network can tolerate multiple number of faults which is up to the minimal critical fault. If number of faults is more than minimal critical faults, the network will be divided into multiple isolate parts and no routing method can connect the terminals belongs to different group. We have designed a network routing algorithm which is applicable to not only fault-free networks but also faulty networks. Finally, we analyzed the performance of our design in various faulty conditions by simulating the faulty networks.

Now we can propose the directions for the future works. First, the fault diagnosis method we proposed here can be improved further. To optimize in time, a pure on-line fault diagnosis method is the most desirable, but it requires more hardware and a large number of control lines (i.e. IC pins). Also the implementation might not be practical. To optimize in space, a pure off-line test is the most desirable without any needs of extra control lines, but this method will severely degrade system performance by delaying network operation. Therefore we developed a combined method which is combination of the pure on-line and pure off-line methods. But we believe this combined diagnosis method still can be improved for the best performance with more investigations of current chip technology.

Second, we believe the proposed routing algorithm still can be improved with the help of our network simulator. For example, some decision points such as the activation point of intermediate algorithm can be modified for better performance.

Third, the interconnection network simulator which we have developed has several segments to accomodate various needs. Since our primary use of the simulator is to gather statistical data for our research, some parts of the simulator still need to be modified for the general purpose user-friendly interconnection network simulator.

Fourth, we suggest that the extension of fault model might provide a good future research work. But since we are dealing with multiple faults environments, it might not be a manageable research task, if you over extend the

fault model. Therefore we suggest that either by limiting the number of faults or considering only few valid faulty states might be a good first approach. Since the nature of this extension is beyond the scope of this report, the investigations for this research will be with the current fault model.

3.6. References

- [1] D.P. Agrawal, "Testing and Fault-tolerance of Multistage Interconnection Networks," IEEE Computer, V-15, pp. 41-53., Apr, 1982.
- [2] Y. Chang and T. Feng, "Interconnection Networks and Their Fault Diagnosis Methods," Technical Reprot TR-86-009, Computer Eng. Program The Penn. State Univ., 1986.
- [3] T.Feng andn C. Wu, "Fault Diagnosis for a Class of Multistage Interconnection Networks," IEEE Transactions on Computers vol. C-30, no. 10, Oct. 1981, pp. 743-758.
- [4] T. Feng, "A Survey of Interconnection Networks," IEEE Computer, Dec. 1981, pp. 12-27.
- [5] D. Lawrie, "Access and Alignment of Data in an Array Processor," IEEE Transactions on Computers, vol. C-24, no. 12, Dec. 1975, pp. 1145-1155.
- [6] S. Pakzad and Y. Chang, "Routing Algorithm for Faulty Unique-path Interconnection Network," Proc. Int. Conf. Computer Appl. in Design, Simm. and Analysis, Fev. 1988, pp. 87-91.
- [7] S. Pakzad and S. Lakshmivarahan, "Interconnection Networks and Fault Tolerance," Journal of Computer Systems Science & Engineering, vol. 3, no. 2, April 1988.
- [8] J.P. Shen, "Fault Tolerance Analysis of Several Interconnection Networks," Proc. 1982 Int. Conf. Parallel Proc, Aug. 24-27, 19892, pp. 102-112.
- [9] S. Sowrirajan and S.M. Reddy, "A Design for Fault Tolerant Full Connection Networks," Technical Report, University of Iowa, 1980.

CHAPTER 4.

HIGH PERFORMANCE VLSI PROCESSOR ARCHITECTURES

4.1. Introduction

Computers have a widespread range of applications in our modern society. High performance computation is required for the solution of the complex problems given in Table 4.1. To cope with these demands technological advances have provided increases in device speed and reliability as well as reductions in hardware cost and physical size. As a consequence of these improvements, it is actually possible to build powerful processors in a single integrated circuit. In spite of this progress, processing speed has not increased sufficiently to solve the complex problems given in Table 4.1. To cope with these demands technological advances have provided increases in device speed and reliability, as well as reductions in hardware cost and physical size. As a consequence of these improvements, it is actually possible to build powerful processors in a single integrated circuit. In spite of this progress, processing speed has not increased sufficiently to solve the complex problems of Table 4.1 in a reasonable time. Therefore, additional improvements must be based on advanced computer architectures and processing techniques.

Advanced computer architectures are centered around the concept of parallel processing. A broad classification of these architectures include pipelined computers, array processors, and multiprocessor systems. Some examples of parallel computer systems are given in Table 4.2. The advent of VLSI has made possible the implementation of multiprocessors in a single chip, and it has been predicted that VLSI devices with millions of gates will be available within the next decade [1,2]. Extensive research is actually carried out to define multiprocessor architectures that capitalize on the high density of VLSI chips. It is a well accepted fact that the performance of a multiprocessor is highly dependent not only on the structure of the system as a whole, but also on the performance of the individual system components such as processors and interconnection networks. The objective of this research is to determine novel VLSI functional processor architectures which can provide high performance in dynamic environments, and make optimum use of the increased density of new VLSI devices.

Table 4.1

High Performance Computer Applications

Predictive Modelling and Simulations

- 1) Numerical Weather Forecasting
- 2) Oceanography and Astrophysics
 - a) Climate predictive analysis
 - b) Fishery management
 - c) Ocean resource exploration
 - d) Coastal dynamics and tides
- d) Socioeconomics and Government use
 - a) Socioeconomic models
 - b) Crime control

Engineering

- 1) Finite Element Analysis
- 2) Computational Aerodynamics
- 3) Artificial Intelligence and Automation
- 4) Remote Sensing Applications

Energy Resources Exploration

- 1) Seismic Exploration
- 2) Reservoir Modeling
- 3) Plasma Fusion Power
- 4) Nuclear Reactor Safety

Medical, Military, Basic Research

- 1) Computer Assisted Tomography (CAT)
 - 2) Genetic Engineering
 - 3) Weapon Research and Defense
 - a) Multiwarhead nuclear weapon design (Cray 1)
 - b) Simulation of atomic weapon effects by solving hydrodynamics radiation problems (Cyber-205)
 - c) Intelligence gathering, such as radar signal processing on the associative processor for the antiballistic missile program (PEPE)
-

Table 4.2

Examples of High Performance Computers

Array Processors

Bit-Slice

Staran
MPP
DAP

Word-Slice

ILLIAC IV
PEPE

Pipelined Computers

Vector Supercomputers

IBM 360/91
CDC 6000/7000 series
STAR 100
TI-ASC
CRAY-1
CYBER-205
VP-200

Attached Processors

AP-120B (FPS-164)
IBM 3838
MATP

Multiprocessor Systems

C.mmp, Cm*
S-1 system
CDC Cyber-170
Honeywell 60/66
DEC System 10
IBM 370/168
IBM 3081
CRAY X-MP

(Carnegie-Mellon University)
(Lawrence Livermore National Lab)

First some well known classifications of computer systems are reviewed in order to obtain a better understanding of the architectural features affecting performance. Then the fundamental computational models, on which the development of new architectures is based, are presented. The selection of the computational model is a crucial step in the design of a computer system. The limitations of the use of parallelism and concurrency in general purpose computer systems is discussed. Particular emphasis is given to those architectural features which improve scalar performance and consequently reduce the imbalance generally found between the high-speed (parallel) and low-speed (serial) modes of operation of a computer system. We then deal with the access overhead associated with a computational task. An increase in computer performance can be obtained by reduction of this overhead. Two architectural approaches that can achieve high-performance by reducing the access overhead are presented: the systolic and the CPU-MRU (memory reconfiguring unit) architectures. An architectural design style which departs from the conventional von Neumann approach is presented: the decoupled access/execute architecture. This approach achieves performance through concurrency of operation of the access and execute processes associated with a computational task. The two main design styles for VLSI uniprocessors, RISC and CISC machine are analyzed, with the objective of determining those architectural features that lead to high performance. Finally, a new architectural design style, the Dynamically Alterable Access Primitive Architecture (DALAP) is presented. Particular emphasis is given to the design criteria and motivations for the architecture.

4.2. Dynamically Alterable Access Primitive (DALAP) Architecture

Proponents of Reduced Instruction Set Computer (RISC) machines generally argue that Complex Instruction Set Computer (CISC) architectures execute code inefficiently mainly because only a subset of the instruction set is used in most applications, and the overhead due to the implementation of complex instructions reduces the performance of the machine as a whole. RISC philosophy advocates the implementation of the most frequently used instructions in hardware, relying on software routines for the execution of the most complex ones. However, important changes in processor architecture and software development have to take place in order to support the RISC design style.

Among the most important ones we can mention: 1) hardware implementation of the processor control unit; 2) efficient register allocation; 3) extensive use of local memory; 4) efficient pipeline scheduling; and 5) use of optimizing compilers.

In this work it is contended that RISC design style is just one of the possible alternative design styles that can be adopted in order to obtain high performance in VLSI processors. In fact, RISC And CISC architectures represent the extremes of a multidimensional design space characterized by limited resources (real estate).

A new architectural design style is proposed: the Dynamically Alterable Access Primitive (DALAP) architecture. The design style is intended to satisfy the following criteria:

- 1) Advances in VLSI technology offer the possibility of manufacturing highly dense devices (in the order of millions of transistors). Efficient use of these resources (real estate) is crucial in the attainment of high performance for newly developed architectures.
- 2) Memory management functions in current large-scale computations require considerable processor time. Increases in processor performance can be achieved by reducing the access overhead and by obtaining concurrency in the access/execute processes associated with a computation task.
- 3) Adaptability of the architecture to match the characteristics of the algorithm being implemented is a necessary requirement to be satisfied by a processor intended to work in a dynamic environment.

The DALAP architecture represents a generalization of the CPU-MRU concept introduced in previous sections. The MRU can be implemented as part of the memory integrated circuit, as shown in Figure 4.1, or as part of a VLSI processor, as illustrated in Figure 4.2.

The MRU-memory alternative of Figure 4.1 can be described as an intelligent memory, i.e. a memory device with processing capabilities mainly oriented to data management operations. The intelligent memory configuration offers the following advantages:

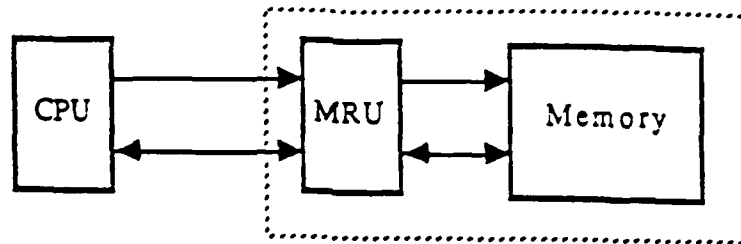


Figure 4.1. Intelligent Memory (MRU-Memory) Configuration

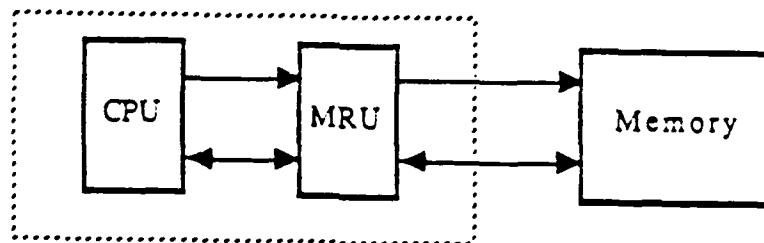


Figure 4.2. CPU-MRU Configuration

- 1) Distributed processing. Several intelligent memories attached to a single CPU can simultaneously carry out the management of the different data structures handled by a computation task.
- 2) Concurrent access/execute operation. The PU can be processing data while the intelligent memories are either retrieving new data or storing partial or final results in memory. This also alleviates the adverse effects of the Flynn's limit.
- 3) Efficient use of the increased density offered by VLSI technology.
- 4) Execution of address sequences calculations on chip reduces the adverse effects of the limited CPU memory bandwidth (von Neumann bottleneck).

The main disadvantage of the intelligent memory approach is that status information, required for the coordinated operation of the CPU and the intelligent memories, has to contend with instructions and data for the available CPU-memory bandwidth. This side effect can drastically reduce performance in a dynamic environment.

A Block diagram of the DALAP architecture is given in Figure 4.3. The architecture consists of two main components: an execute processor (EP), dedicated to data manipulation operations, and an access processor (AP), for data management.

The instruction set of the EP is restricted to operate on data held in registers D_0 through D_m . EP also have read/write access on the registers of the access processor. No addressing modes are associated with the EP instruction set. Consequently, the EP can be considered as a load/store architecture. The absence of complex addressing modes simplifies the EP instruction repertoire considerably allowing the implementation in hardware of its control unit.

The access processor in the DALAP architecture of Figure 4.3 includes four sets of registers:

Mode registers $\{M_0 \dots M_N\}$

Base registers $\{B_0 \dots B_N\}$

Relative displacement registers $\{R_0 \dots R_N\}$

Value registers $\{V_0 \dots V_N\}$

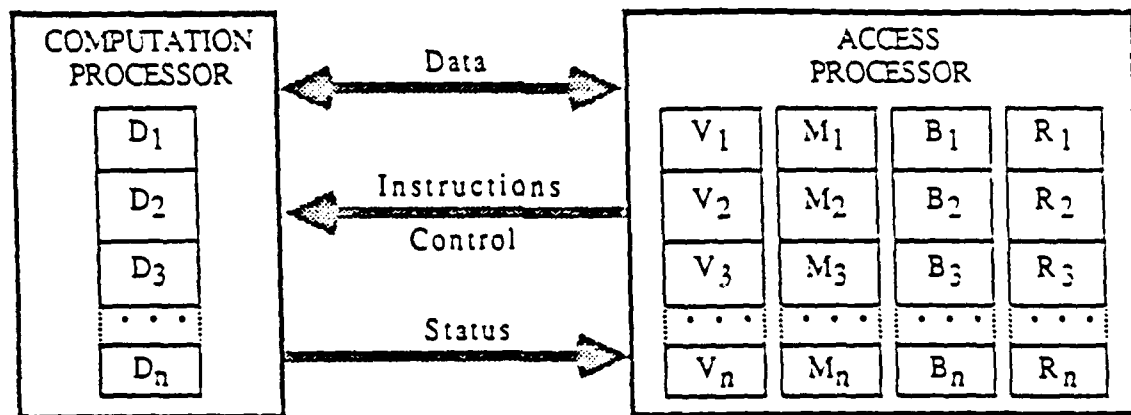


Figure 4.3. Dynamically Alterable Addressing Primitive Architecture (DALAP Architecture)

At this point, it is important to emphasize that the block diagram of Figure 3 describes an emerging architectural concept. Consequently, changes in its architectural features might occur as a result of future research. For example, new sets of registers could be added to the AP, or the existing ones could adopt different functions. The DALAP architecture of Figure 4.3 is used just as a vehicle to illustrate the main ideas supporting the new design style.

4.3. Advantages of the DALAP Architecture

The DALAP design style lies on the multidimensional design space flanked by the RISC and CISC approaches. Indeed, it reflects characteristics of these two architectural design styles. The execute processor resembles, to a certain extent, a RISC machine. Though the accessing primitives implemented in the access processor, the system is endowed with powerful addressing modes, which generally characterize a CISC processor. Due to the decoupling of the access and execute mechanisms, and the implementation in hardware of the addressing primitives, the access overhead normally associated with a computation task is reduced considerably. In addition, by alteration of the addressing primitives characterizing the access processor, the system can be tuned to the application. Finally, the DALAP design style illustrates an attempt to make efficient use of the additional resources (real estate) offered by the advances in VLSI Technology.

All these characteristics lead us to believe that the DALAP architecture can offer high performance in dynamic computational environments.

4.4. Conclusions

Advances in VLSI technology have caused dramatic improvements in the performance of computer systems. Increases of performance beyond the limits imposed by technology can be achieved through the use of concurrency and/or parallelism in the design of the underlying architectures. A reconfigurable multiprocessor is an ideal candidate for large-scale computations in dynamic environments. Reconfigurability allows the system to be adapted to the application so that high performance is maintained throughout the spectrum of algorithms being implemented in the computer system. The availability of multiple processors offers the possibility of achieving fault tolerance and

graceful degradation. These characteristics are required for the solution of battle management algorithms in spaceborne computer systems.

Every computer system endowed with parallel processing capabilities exhibits a high-speed and a low-speed modes of operation. The high-speed mode is active whenever the computation task can be optimally distributed among the processing elements. However, those parts of the algorithm being implemented that do not exhibit parallelism can force the machine to behave as a single processor. Under these circumstances the machine is said to be operating in the low-speed mode. To obtain high performance the system must operate efficiently in both modes. Therefore, the design of the individual processors becomes as important as the design of the architecture of the whole system. This fact is supported by ample experimental evidence obtained the analysis of many computer systems with parallel processing capabilities, and it is known in the literature as Amdahl's law. As a direct consequence, research on high performance VLSI uniprocessors becomes a crucial step in the design of high performance multiprocessors.

The von Neumann's bottleneck and the Flynn's are two main limitations hindering performance in VLSI uniprocessors. Special purpose devices, as for example systolic and wavefront processors, have been proposed to alleviate the aforementioned limitations. The special purpose architectures offer high performance for very specific applications. They are commonly used as peripheral processors attached to general purpose computers. For large-scale computations in dynamic environments, architectures which can offer sustained high performance for wide class of applications are highly desirable. Current architectural design styles that take advantage of the increased density offered by VLSI technology are: complex instruction set computers (CISC) and reduced instruction set computers (RISC). RISC and CISC machines are at the extrema of a multidimensional design space with limited resources (real estate). An analysis of the RISC-CISC controversy is carried out in order to determine those architectural features that can lead to high performance in a dynamic computation environment. Particular emphasis is given to the architectural support for memory management functions. Most large-scale computer applications are developed using high level procedural languages and operate on data structures. Experimental evidence has revealed that a considerable amount of processing time is devoted to the generation of the address sequences aimed at reducing the access overhead of computation tasks, based on the concept of memory

reconfiguration unit (MRU), is presented. The MRU can be implemented as part of a VLSI Memory chip. This leads to the concept of intelligent memory which offers the advantages of distributed processing and concurrent access/execute operation. However, the flow of status information required for the efficient interlock between the CPU and the intelligent memory system has to compete for the limited CPU-memory bandwidth (von Neumann bottleneck). This can cause dramatic losses of performance. A better alternative relies on considering the implementation of the MRU as part of the VLSI processor. This scheme allows concurrency in the access and execute processes and handling of status information inside the chip. A generalization of this concept leads to the Dynamically Alterable Primitive Addressing (DALAP) architectural design style. A DALAP architecture consists of an execute processor (EP) and an access processor (AP), which operate concurrently in the solution of a task. The design of the EP is based on RISC principles, and the design of the AP supports the implementation of a set of addressing primitives which can be selected for the application. The DALAP design style offers the following main advantages: 1) decoupled access/execute operation; 2) adaptability by means of access primitive selection; 3) efficient use of the high density offered by VLSI technology; 4) efficient memory management functions; 5) reduction of the access overhead associated with computational tasks. These characteristics underlying the DALAP design style point to the attainment of high computational performance.

Future research efforts are recommended at:

- 1) Defining the architectural features characterizing the access and execute processors of the DALAP architecture
- 2) Determining the set of addressing primitives according to the characteristics of the algorithms being implemented
- 3) The architectural features that are required in order to incorporate the DALAP architecture in a multiprocessor
- 4) Optimal use of the available resources (real estate) in the implementation of the DALAP architecture.

4.5. References

- [1] Barbe D.F., and Urban E.C. "VHSIC Technology and Systems." VLSI Fundamentals and Applications. 1982. pp. 255-271.
- [2] McGreivy D.J., and Pickar K.A. "VLSI Technologies Through the 80s and Beyond." IEEE Computer Society Press. 1982.

CHAPTER 5

VLSI BUILT-IN SELF TEST

5.1. Introduction

As Integrated Circuit technology has progressed from Large Scale Integration (LSI) to Very Large Scale Integration (VLSI) there has been a decrease in gate costs by as much as three to five times, along with an improvement in performance. However, a problem never adequately solved for LSI is still present and is getting worse. This is the problem of determining a cost-effective way whether a component, module or board has been manufactured correctly, and when incorporated in a system is functioning correctly.

Testing is a significant portion of the design cost, its effect being felt all the way to the field application of the system. A standard among test engineers is that if it costs \$0.30 to detect a fault at the chip level, it costs \$3 to detect it at the board level, \$30 at the system level and \$300 if it is found in the field. It has been predicted that in the future the key areas in VLSI testing, where test problems are likely to occur, are as follows:

1. Test Cost. Test cost is proportional to the test time and test time for VLSI circuits is increasing faster than gate complexity. Unless some solution is found to this problem, test cost per circuit in VLSI will be greater than in LSI.
2. Test Pattern Generation Cost. Computer time required for test generation is also increasing faster than circuit count.
3. Test Data Volume. Test data is growing faster than either test time or test generation time. Not only is storage and transmission a problem but this also results in a more expensive and slower tester.
4. Tester complexity. There is an increase in the complexity of the tester as the number of I/O pins increases and quantity of test data grows. This will again contribute to increased test cost.

Thus, there is little question that the host of opportunities VLSI offers for future advances in electronic systems can be realized in practice only if the major bottleneck of time and cost required for testing can be overcome.

5.2 VLSI BIST Techniques

Numerous techniques are currently available for providing Built-in Self-Test (BIST) in VLSI designs. There are many choices for the circuit designer and each technique has merit in certain design situations. A criterion of choice can be outlined based on design objectives. It is more difficult to obtain a measure of comparing possible alternatives. The work described in the accompanying report involves classification and comparative evaluation of existing techniques for VLSI-BIST. The evaluation parameters include hardware overhead, pin overhead, possibility of extension to system level self-test, system performance, and test time. This study will help self-test tradeoff decision making and will aid designers in incorporating self-test features in their designs. The results of this study are directed towards well-orchestrated board and system level BIST approaches based on core chip capability.

5.3 Application of BIST Techniques

Studies are made for applying the BIST techniques to digital systems which are suitable for high performance fault-tolerant battle management applications. Extensions of these techniques to the general heterogeneous system is shown to be both feasible and desirable; using the proposed system self-test strategy. Relevant performance issues in mission critical fault-tolerant C^3 applications are identified. Compact built-in self-test techniques are proposed to support such systems, and alternate self-test architectures are identified. Results include modeling the effects of BIST techniques on system computation availability and area utilization. A comparison of the Penn State approach with a commercial test/self-test methodology is also present.

This work develops the basis for automatic or computer-assisted support of comprehensive hardware diagnostics and fault tolerance in large distributed military systems. Furthermore, this approach produces results which are applicable to the development, manufacturing, repair depot, and fault tolerant field environments. An additional advantage is that this research is directed towards producing a generalized approach to self-testing system design. This approach results in a uniform design strategy which is applicable to any new system start. Thus significant economies of scale can be realized over conventional system design techniques wherein a unique system architecture/hardware design is developed for each specific problem application.

CHAPTER 6

A STATE OF THE ART RECONFIGURABLE DATABASE MACHINE

6.1. Introduction

6.1.1. Reconfigurability

Since the last decade, conventional database systems have become more complex due to the constant growth in the number and size of databases on one hand and the number and type of on-line users on the other hand. It has been shown in practice that the conventional von-Neumann type architecture is not well suited for database applications because of its: i) inherent sequential control flow nature, ii) orientation for numeric operation and iii) passive address accessible memory organization. These characteristics differ from the content addressability and non-numeric nature of the database operations. As a result, the existing application gap and its natural byproducts (i.e. computation gap, data communication problem and name mapping resolution problem) have been a driving force in the introduction and continuous research of database machines since the early 70's.

The majority of the recent database machine proposals advanced in the literature can be classified as either multiprocessor or multicomputer systems. Such a direction is due to the faster response time and higher throughput of these systems in a multiprogram/multiuser environment. However, such an environment introduces issues such as: allocation of processors to concurrently running queries, increased overhead due to data/control communication among processors, increased control overhead, and concurrency control; these issues should be studied in greater depth. It should be noted that because of the current and foreseeable advances in technology and its effect on reducing the cost, size and switching delay, such a direction in the design of database machines is both reasonable and feasible.

Since the third generation computer era, the dynamic distribution of the hardware resources in a parallel system among concurrently running tasks has been a viable solution to enhance the system's resource utilization, throughput, availability and reliability, and fault-tolerant capability. Resource reconfigurability has been incorporated at different levels of the architecture in the past. For example, TI-ASC [1], RP3[2] and TRAC [3] allow the dynamic

reconfigurability of the resources at ALU, processor, and memory levels respectively. In addition, the Michigan database machine [4] allows a degree of reconfigurability by exercising different algorithms based on the specific characteristics of the database involved in the operations.

We believe that due to the general characteristics of databases one can develop a set of simple algorithms in order to enforce dynamic allocation of the resources among different independent queries in a multiuser environment. The complexity of such algorithms are clearly architectural dependent and a function of such parameters as: complexity of the queries, number of tuples in a database, size of the datablocks, data transfer rate among modules, sensitivity of a datablock to a query, etc. We have developed such a set of algorithms which allows dynamic distribution of the resources in the database machine ASLM among concurrently running queries. Interestingly, dynamic reconfigurability of the hardware resources enhances the fault tolerance capability of the underlying architecture, since the faulty elements can be bypassed rather than halting the operation.

Study and analysis of the developed reconfigurable algorithms in the proposed database machine was one of the major themes of this research activities.

6.1.2. Incomplete Data

The problem of how to handle missing information has been studied by a number of researchers [5-15]. A number of different uses for null values have been given in [16-17], but the problem of unknown data value represents the most common usage of null values. Codd [6] introduced a comprehensive extension to relational algebra that allows the user to examine potentially interesting data relationships based in part on unknown (incomplete) data. the theoretical foundations of Codd's new maybe algebra have been examined by Biskup [18].

While these works have developed a strong theoretical basis for maybe algebra, they have not taken into consideration the practical impact of such operations. Moreover, the incorporation of these operations (e.g. Codd's maybe algebra operators [6-7]) have not generally been addressed in the design of the database machines.

The incorporation of the concept of null values and maybe algebra operators in the design of the proposed database machine was the second objective of our activities.

6.1.3. Database Integrity and Consistency

The relational model makes use of a dependency set to describe the real world semantics of the data. Three types of dependencies have generally been considered in database design theory - functional, multivalued, and join dependencies. In this work, we restrict the dependency set to include only functional dependencies. While the other dependency types may impact the integrity of the database, enforcing them require multiple levels of testing. Such extra cost does not appear to be reasonable even in the parallel environment provided by database machines. In addition, by restricting our view to relational databases we are omitting the integrity constraints available in the logical database environment.

The functional dependency is defined by Ullman [14] as:

X functionally determines Y (denoted $X \rightarrow Y$) if it is not possible that two tuples agree in components for all attributes in X and disagree on the components for the attributes in Y.

It is clear that such an integrity constraint can be tested by comparing one tuple against the set of tuples that comprise the relation.

Codd denoted that a relation that has been defined without considering the role of the functional dependencies may demonstrate some update anomalies. The basic problems can be identified as

- i) data inconsistency - updating a portion of a set of redundant data will create a problem of inconsistent data fields.
- ii) insertion anomaly - relevant data items may be blocked from being stored in the database if important elements of a tuple are not available.
- iii) deletion anomaly - relevant data may be deleted by the deletion of the last tuple containing the data.

To combat the anomalies, Codd introduced the concept of normalization where the first normal form (1NF) places the data in a tabular format. Codd introduced the second and third normal forms to reduce the effect of the update anomalies. In particular, the two normal forms were intended to remove non full functional dependencies (in $X \rightarrow Y$, the attributes of Y must be determined by all the attributes of X to be full functional dependency) and non-key transitivity, respectively. Boyce-Codd Normal Form (BCNF) has been introduced to deal with anomalies caused by overlapping keys. While the normal forms were introduced to improve the logical performance of the data model, they also have an impact on testing integrity constraints. One obvious issue is that higher levels of normal forms tend to reduce relation size which may result in some dependencies no longer being defined with one relation scheme.

Databases fall into two general categories, namely formatted and unformatted structures. Formatted databases are mainly time variant entities and subject to extensive alteration as well as search operations. Unformatted databases (bibliographic or full text) are archival in nature and are processed by searching for a pattern or a combination of patterns.

Since formatted databases are subject to continual changes during the life time of a system, the database management system should provide facilities to allow modification operations as well as retrieval operations. Among these two classes of operations, modification imposes a higher degree of complexity. This complexity is due to the:

- i) Data dependence between the concurrent execution of the modification operations on one hand, and modification operations and retrieval operations on the other hand,
- ii) Requirement for proper algorithms and procedures to guarantee the data integrity and consistency, and
- iii) Requirement for proper recovery procedures in order to recover the original data in case of a system malfunction or improper (intentional or accidental) data manipulations.

In general, to guarantee system integrity and data consistency, a database system:

- i) Should not allow concurrent execution of data dependent operations by imposing proper locking protocol during the operations, and
- ii) Should enforce the content/context dependent integrity constraints during the execution of the modification operations.

A Comprehensive policy for using functional dependencies to improve integrity for ASLM was the third objective of our efforts.

6.1.4. Temporal Databases

In the past few years, a number of researchers have looked at the question of providing temporal support as part of the data model. McKenzie [26] has compiled a comprehensive bibliography of the recent work on temporal databases. Snodgrass [28] profiles the active research programs looking at some aspects of time in the database environment. Current results have gone a long way towards incorporating the required support into the data model.

Maintaining a historical database exasperates the problems of operating on large data collections. The projected size of the historical databases makes it seem unlikely that traditional software approaches will be adequate. Lum et. al [29] have proposed a transaction time system which partitions the database into current data and historical data. Such an approach preserves the traditional operations on the current database, but their approach is based on the use of indexing and as such suffers the limitations of the software solution.

The feasibility of providing temporal support in database machine ASLM and its application in the decision making process was the final objective of our attempts.

6.2. Results

6.2.1. Hardware Reconfigurability

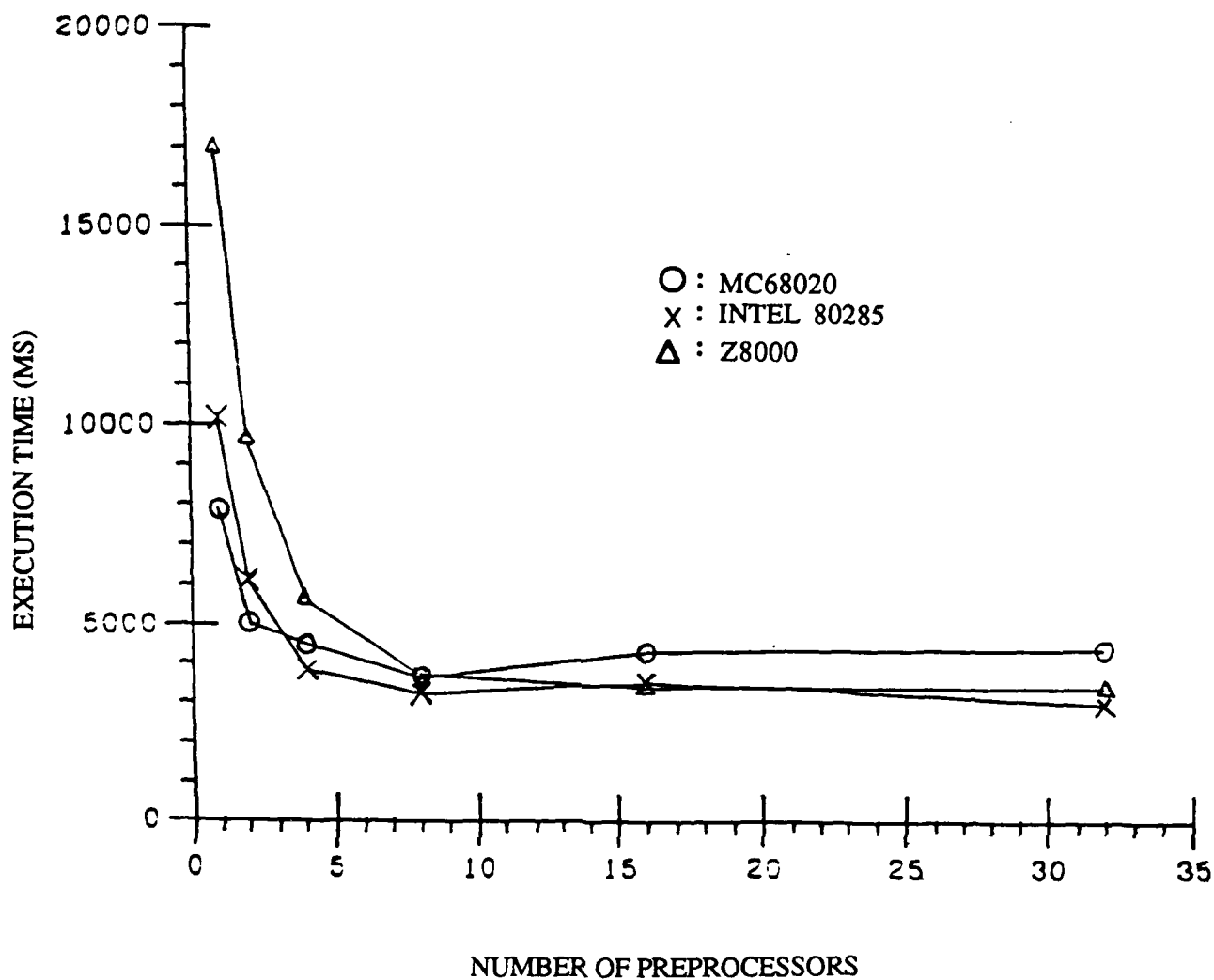
The modular design, replicated nature, and hardware independence of the components of ASLM allowed us to develop a sequence of control algorithms to enhance the use of the machine resources. The single user environment of ASLM described in the previous section does not lend itself to a high level of hardware utilization. Features such as the horizontal and vertical concatenation of the associative modules allows the resources to be adjusted according to the user's query at the point of query initiation. A more appropriate policy would allow reallocation of the hardware resources within a component on a dynamic basis. It

is clear that the dynamic reconfigurability of ASLM is closely tied to the extension of ASLM from a single user/single program environment to a multiuser/multiprogram environment. As a result, one should expect a higher throughput and performance. Moreover, dynamic reconfigurability of the hardware resources enhances the fault tolerance capability of ASLM since the faulty element can be bypassed rather than halting the operation. We have developed a set of algorithms which allows the dynamic allocation and deallocation of secondary storage interface cells, preprocessors and associative modules among a set of atomic queries (an atomic query is referred to the request of loading a relation).

Average Execution Time

As one can expect, ASLM performance is closely related to the performance and capability of the preprocessors. Entries in Figure 6.1 shows the average execution time of the join operation for various type and number of preprocessors. As can be concluded, for lower number of preprocessors, faster units offer a better performance. However, as the number of preprocessors increases this relationship does not hold anymore. This anomaly is contributed to the facts that:

- Faster preprocessors implies simultaneous execution of higher number of queries. As a result, one should expect a resource contention due to the higher demands for the communication lines and access to the backup resources (e.g. secondary storage interface). This so called resource contention increases the preprocessor idle time and hence, increases the average execution time.
- For higher number of preprocessors (especially for faster units) the probability of simultaneous execution of several subqueries on the same relation increases. As a result, due to the synchronization policy as discussed before, the preprocessor idle time increases, which results in an increase in the average execution time.



- Each entry is an average of 150 simultaneous runs
- Number of auxiliary storage = 2 units
- Number of communication lines between preprocessors and secondary storage interface = 2
- Number of communication lines between secondary storage and secondary storage interface = 2
- Join selectivity = 1
- Preprocessing selectivity = 0.1

Figure 6.1. Average Execution Time of the Join Operation for Different Types and Number of Preprocessors

Overall Performance

Entries in Figure 6.2 shows the total execution time of the join for various type and number of preprocessors. As the figure shows, faster preprocessors offer a better performance. In addition, an increase in the number of preprocessors will improve the performance.

Effect of the Auxiliary Memory

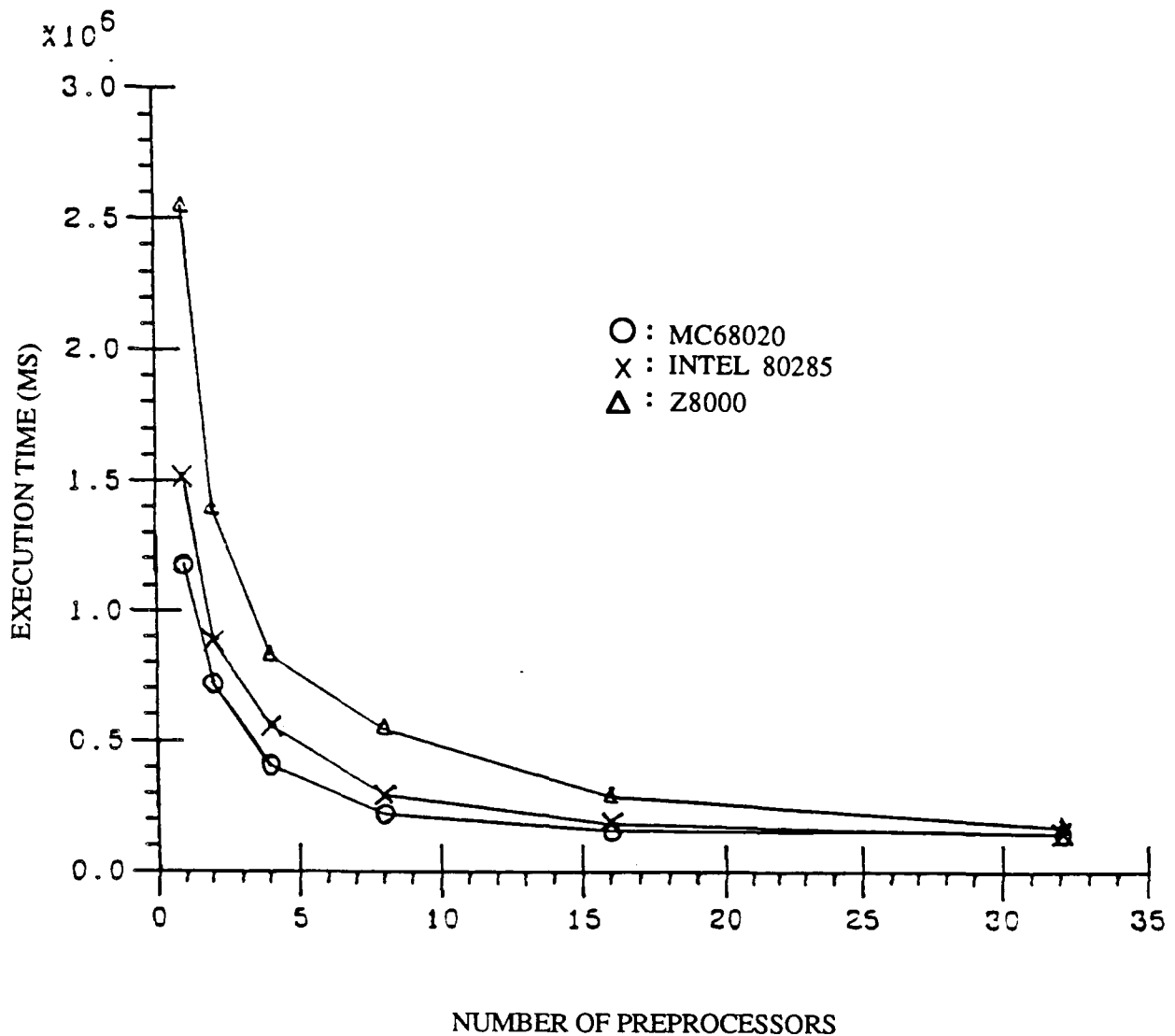
Our analytical evaluation showed that the overall performance of the ASLM can be improved if preprocessors are backed up by a fast access memory. Figure 6.4 depicts this fact. As can be seen by increasing the parallel access capability of the auxiliary storage the average execution time drops.

Parallelism

As one could expect, in a parallel system an increase in the number of resources should increase the degree of parallelism and hence one should expect a better performance. Figure 6.3 shows this fact, indeed by increasing the number of preprocessors the ability of concurrent execution of queries increases, this has resulted in a higher overall performance for MIMD organization relative to SIMD organization. However, in a separate simulation run, we have shown that the average execution time for the MIMD organization is higher than the SIMD organization. This is due to the complexity of the MIMD model over the SIMD model.

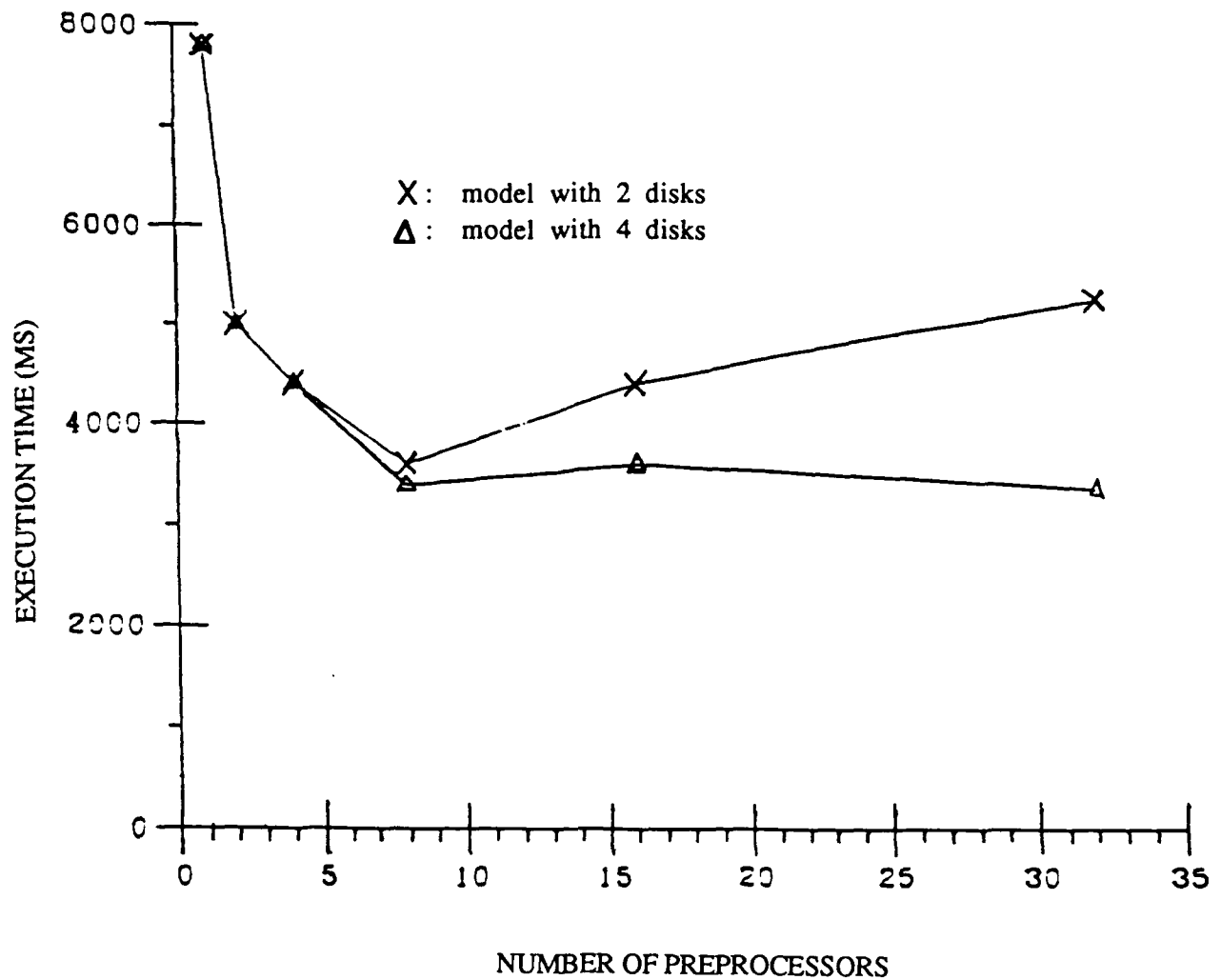
6.2.2 . Handling Incomplete Data in ASLM

As discussed earlier, enhancement of ASLM to handle incomplete data and maybe algebra operators is a major goal of this research proposal. In the remainder of this section, we examine the action of the join module for the four join operations. We start the discussion by looking at some features that are common to all four types of join. In ASLM, the two relations to be joined are loaded into individual associative modules: the source module (S_S) and the target module (S_T). The tuples in S_S are tested one at a time against the tuples in S_T .



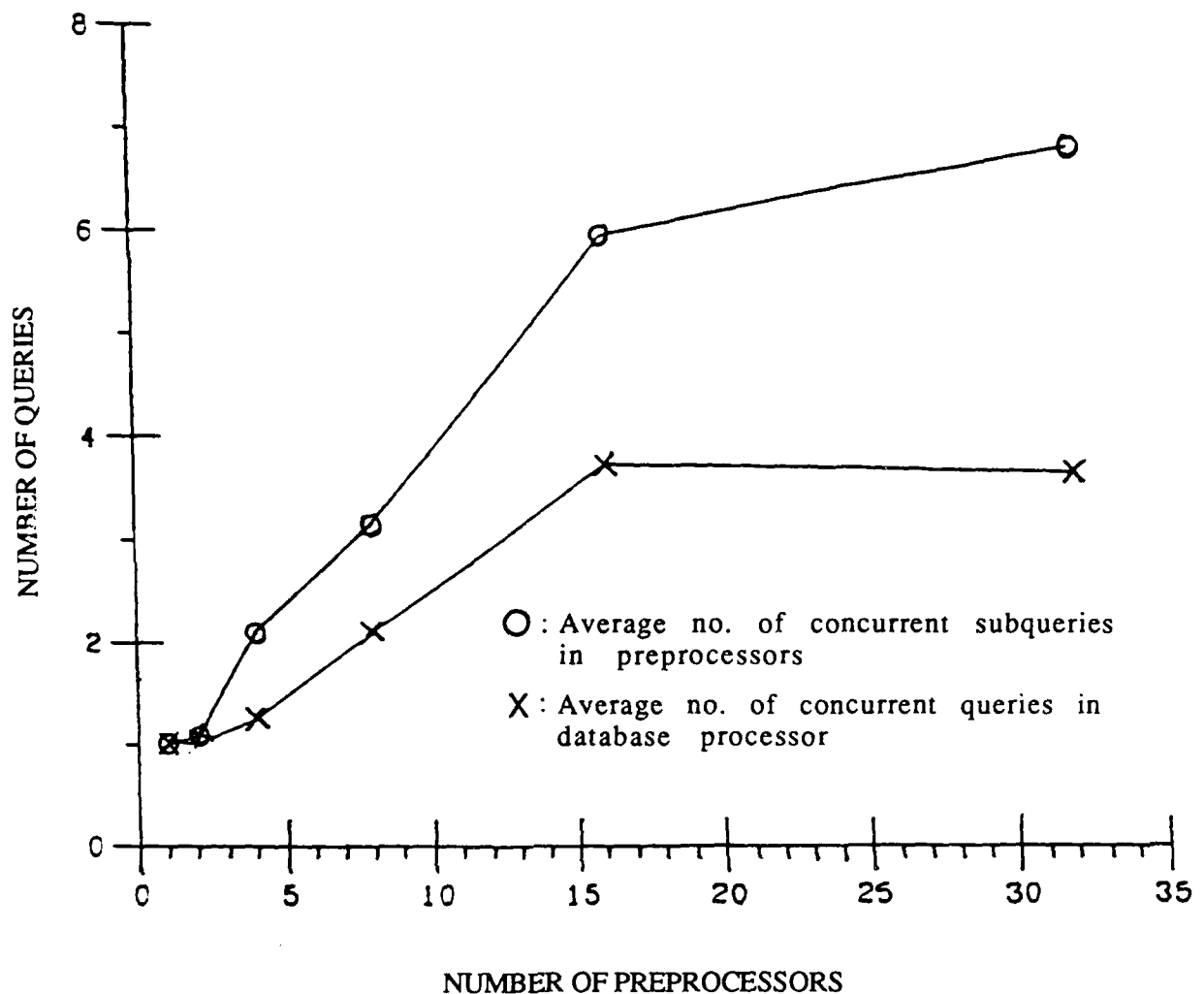
- Each entry is an average of 150 simultaneous runs
- Number of auxiliary storage = 2 units
- Number of communication lines between preprocessors and secondary storage interface = 2
- Number of communication lines between secondary storage and secondary storage interface = 2
- Join selectivity = 1
- Preprocessing selectivity = 0.1

Figure 6.2 Total Execution Time of the Join Operation for Different Type and Number of Preprocessors



- Each entry is an average of 150 simultaneous runs
- Number of auxiliary storage = 2 units
- Number of communication lines between preprocessors and secondary storage interface = 2
- Number of communication lines between secondary storage and secondary storage interface = 2
- Join selectivity = 1
- Preprocessing selectivity = 0.1

Figure 6.3. Average Execution Time vs. the Parallel Access Capability to the Auxiliary Storage



- Each entry is an average of 150 simultaneous runs
- Number of auxiliary storage = 2 units
- Number of communication lines between preprocessors and secondary storage interface = 2
- Number of communication lines between secondary storage and secondary storage interface = 2
- Join selectivity = 1
- Preprocessing selectivity = 0.1

Figure 6.4. Concurrency vs. Hardware Resources

Tuples in S_T that satisfy the join condition for a source relation tuple are joined with the source tuple and placed in a third associative module (the destination module - S_D).

It is natural to assume that one or more of the relations will be too large to allow its being stored entirely in an associative module. To accommodate large relations, ASLM stages the relation between auxiliary memory and the associative module. As previously described, the tuples in S_S are compared one at a time with the tuples in S_T . If the target does not fit in S_T , then the process repeats for each successive load of S_T . When the tuples in S_S have all been tested against the tuples in the target relation, the associative module S_S is loaded with the next set of tuples from the source relation and the process repeats. Since the entire target relation must be loaded for each load of S_S , it is clearly advantageous to use the smaller relation as the target relation. Interestingly, the opposite is true for the case where both relations fit into their respective associative modules.

Analytically and intuitively, several trends were expected in the results. For example, under the same conditions, true join would be expected to have a shorter execution time and smaller resultant relation than the maybe join. The results for attribute maybe join should lie between those for true join and maybe join. Generally, our simulation results bore out these expectations and produced failure constant relationships between the varying parameters and the execution time and size. Table 1 shows some of the results.

6.2.3. Enforcement of Integrity Constraints in ASLM

As noted before, automatic enforcement of the integrity constraints during the modification operations is one of the major themes of this proposal. In our preliminary study for a comprehensive insertion and modification operation, we have considered two general cases, namely Total relations and Partial Relations.

We have developed a comprehensive policy for using functional dependencies to improve integrity for ASLM. The proposed algorithms maintain the performance level of the retrieval primitives while expanding the usefulness of ASLM. In addition, we have calculated the overhead of such an algorithm analytically. Our analysis has shown that the architecture of ASLM has the capacity to improve the integrity of the data while not suffering the extreme costs required in the software approach.

Table 6.1. Relative Performance of the Join Algorithms

	Resultant Relation (Tuples)	C_{join} (μ_s)
True Join	24.3	211.454
Attribute Maybe Join	38.05	564.813
Maybe Join	57.75	850.807
Outer Join	121.85	222.628

relation size = 50 blocks

join attributes = 3

Each entry is the average of 200 simulation runs.

6.2.4. Handling Time in ASLM

As reported in [37,38], we have proposed a query language - ASL (Associative Search Language) to allow easy communication between user and ASLM. We examined the conversion of the ASL to Temporal ASL (TASL) by incorporating time related clauses into the language.

We have shown that ASLM can be upgraded to support time primarily by modifying the query language and correspondingly upgrading the compiler. Moreover, the support has been added in such a way so as to maintain the level of performance for traditional processing. Since the current data queries are processed using only the current file, the performance remains essentially the same. The only apparent change is that tuples carry the extra fields and as such the amount of data transferred increases somewhat.

6.3. References

- [1] Watson, W.J., "The TI-ASC A Highly Modular and Flexible Super Computer Architecture," Fall Joint Computer Conference, 1972, pp. 221-229.
- [2] Pfister, G.F., et. al., "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture," Int. Conf. on Parallel Processing, 1985, pp. 764-771.
- [3] Deshpande, S.R., Jenevein, R.M., and Lipovski, C.J., "TRAC: An experience with a novel architectural prototype," National Computer Conference, 1985, pp. 247-258.
- [4] Qahad, G.Z. and Irani, K.B. "A Database Machine for Very Large Relational Databases," IEEE Transactions on Computers, Vol. C-34, No. 11, 1985, pp. 1015-1025.
- [5] Codd, E.F., "Understanding Relations," EDT, Vol. 7, No. 3-4, December 1975, pp. 23-28.
- [6] Codd, E.F., "Extending the Database Relational model to Capture More Meaning," ACM Transactions on Database Systems, Vol. 4, No. 4, 1979, pp. 397-434.
- [7] Codd, E.F., "Missing Information (Applicable and Inapplicable) in Relational Databases," SIGMOD Record, Vol. 15, No. 4, 1986, pp. 53-78.
- [8] Date, C.J., "Null values in Database Management," Proceedings of the 2nd British National Conference on Databases, Bristol, England, July 1982; Also Chapter 14 in Date C.J., Relational Databases: Selected Writings, Addison-Wesley, 1986.
- [9] Grant, J., "Null Values in a Relational Database," Information Processing Letters, October 1977, pp. 156-157.
- [10] Grant, J., "Partial Values in a Tabular Database Model," Information Processing Letters, August 1979, pp. 97-99.
- [11] Lien, Y.E., "Multivalued Dependencies with Null Values in Relational Databases," Proceedings of Very Large Databases, Brazil, 1979, pp. 61-66.
- [12] Lipski, W., "On Semantic Issues Connected with Incomplete Information Databases," Transactoins on Database Systems, Vol. 4, No. 3, 1979, pp. 262-296.
- [13] Lipski, W., "On Databases with Incomplete Information," Journal of ACM, Vol. 28, No. 2, 1981, pp. 41-47.
- [14] Ullman, J., Principles of Database Svstems, (Second Edition), Computer Science Press, Rockville, Maryland, 1982.

- [15] Vassilion, Y., "Null Values in Database Management Denotational Semantics Approach," ACM SIGMOD Conference, 1979, pp. 162-169.
- [16] ANSI/X3/SPARC Study Group on Database Management Systems, Interim Report, ANSI, Feb. 1975.
- [17] Parker, D.S., and P. Atzeni, "Assumptions in Relational Database Theory," Proceedings of the ACM Symposium on Principles of Database Systems, March 1982, Los Angeles, pp. 1-9.
- [18] Biskup, J., "A Foundation of Codd's Relational Maybe-Operations," ACM Transactions on Database Systems, Vol. 8, No. 4, 1983, pp. 608-636.
- [19] Dewitt, D.J., "DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems," IEEE Transactions on Computers, Vol. C-28, No. 6, 1979, pp. 395-406.
- [20] Kitsuregawa, M., Tanaka, H. and Moto-oka, T., "Architecture and Performance of Relational Algebra Machine GRACE," Proc. of Int. Conf on Parallel Processing, 1984, pp. 241-250.
- [21] Fushimi, S., Kitsuregawa, M., and Tanaka, H., "An Overview of the System Software of a Parallel Relational Database Machine GRACE," Proc. of the Twelfth Int. Conf. on Very Large Data Bases, Kyoto, Japan, August 1986, pp. 209-219.
- [22] Baru, C.K. and Su, S.Y.W., "The Architecture of SM3: A Dynamically Partitionable Multicomputer System," IEEE Transactions on Computers, Vol. C-35, No. 9, 1986, pp. 790-802.
- [23] Dewitt, D.J., Gerber, R.H., Graefe, G., Heytens, M.L., Kumar, K.B. and Malikrishna, M. "GAMMA-A High Performance Dataflow Database Machine," Proc. of the Twelfth Int. Conf. on Very Large Data Bases, Kyoto, Japan, August 1986, pp. 228-237.
- [24] Gerber, R.H., and Dewitt, D.J., "The Impact of Hardware and Software Alternatives on the Performance of the Gamma Database Machine," Technical Report #8708, University of Wisconsin, Madison 1987.
- [25] Bit, L., and Hartmann, R.L., "Hither Hundreds of Processors in a Database Machine," Database Machines: The Fourth International Workshop, Springer-Verlag, 1985, pp. 153-168.
- [26] McKenzie, E., "Bibliography: Temporary Database," SIGMOD Record, Vol. 15, No. 4, 1986, pp. 40-52.
- [27] Snodgrass, R., "Research Concerning Time in Databases: Project Summaries," SIGMOD Record, Vol. 15, No. 4, 1986, pp. 19-39.

- [28] Snodgrass, R. and I. Ahn, "A Taxonomy of Time on Databases," Proceedings of the International Conference on the Management of Data, 1985, pp. 236-246.
- [29] Lum, V., P. Dadam, R. Erbe, J. Guenour, P. Pistor, G. Walch, H. Werner and J. Woodfill, "Designing Support for the Temporal Dimension," Proceedings of the International Conference on the Management of Data, 1984, pp. 115-130.
- [30] Clifford J. and A.U. Tansel, "On an Algebra for Historical Relational Databases: Two Views," Proceedings of the International Conference on the Management of Data, 1985, pp. 247-63.
- [31] Clifford J. and D.S. Warren, "Formal Semantics for Time in Databases," ACM TODS, Vol. 8, No. 2, June 1983, pp. 214-254.
- [32] Gadia, S.K. and j.H. Vaishnav, "A Query Language for a Homogeneous Temporal Database," Proceedings of the Fourth ACM Symposium on the Principles of Database Systems, 1985, pp. 51-56.
- [33] Snodgrass, R., A Temporal Query Language, University of North Carolina Technical Report, TR85-013, May 1985.
- [34] Snodgrass, R., "The Temporal Query Language TQUEL," Proceedings of the Third ACM Symposium on Principles of Database Systems, 1984, pp. 204-212.
- [35] Hurson, A.R., "VLSI Time/Space Complexity of an Associative Parallel Join Module," Proceedings of International Conference on Parallel Processing, 1986, pp. 379-386.
- [36] Pegden, C. Dennis, Introduction to SIMAN, System and Modeling Corporation, 1986.
- [37] Mukhopadhyay, A., and Hurson, A.R., "An Associative Search Language for Data Management," National Computer Conference, 1979, pp. 727-732.
- [38] Hurson, A.R., and Fu, S.M., "ASL-A Query Language for Database Systems," International Computer Conference, 1984, Vol. 2, pp. 765-774.
- [39] Hurson, A.R., Petrie, C.R. and Cheng, J.B., "A VLSI Join Module," Proceedings of 21st Hawaii Conference on System Sciences, 1988, pp. 41-49.

CHAPTER 7

DISTRIBUTED TRANSACTION PROCESSING

7.1. Introduction

Several factors have motivated the decentralization of computer systems in general and have made distributed systems economically interesting. These factors include the decrease in processor and communication (hardware) costs, the desire for sharing expensive resources (gathered data), the desire for incremental expansion, and the need for applications requiring geographic separation of system components (remote sensing). Distributed computing is becoming more and more common and the need for building reliable systems has become a major area of research. Higher reliability could be achieved by the realization of reliable hardware as well as reliable software. This research addresses the issue of reliable software. One of the software tools that has become a key concept in the field of database is the **transaction**. A transaction is defined as an atomic collection of indivisible interactions with the database that guarantee system consistency. Transactions incorporate recovery schemes to cope with failures, and concurrency control schemes to cope with concurrent executions with other transactions.

Even though transactions were originally developed for centralized database systems, considerable research effort is directed towards extending their utility beyond these applications. In fact, several operating system research groups have already applied the transaction concept to some operating system functions [1, 19, 28, 27].

Since transactions proved to be a very useful notion for implementing reliable software, there has been a growing interest in extending them to the distributed environment. The main area of application of distributed transactions has been that of distributed database systems.

The next two sections briefly mention the motivations for distributed database systems and identify some of the research issues that were raised as a result of this distribution.

7.1.1. Motivation for Distributed DBMS

Database systems, DBMS, first appeared as centralized. This scheme provided the database managers with centralized control of the operational data thus allowing them to reduce redundancy, avoid inconsistency, share the data, enforce standards, restrict security, and maintain integrity.

For database systems, the two main motivations for distribution are the geographically dispersed location of data as well as reliability. For many of today's organizations, it is quite common that database users and data are themselves physically distributed. Therefore, a distributed system will suit the organization's distributed structure more naturally and better than a centralized system. A distributed database system consists, typically, of a collection of autonomous centralized database systems interconnected by a communication network and logically integrated to form a distributed system. This logical integration is achieved by using a distributed database management system, DDBMS.

7.1.2. Research Issues in DDBMS

Distributed DBMSs provide significant advantages over centralized DBMSs such as better reliability, better availability, data sharing, and incremental growth. However, Distributed systems raise new problems that are not present in their centralized counterparts. These problems translate into five major issues to be addressed: Namely, file allocation, catalog management [18], query processing [2, 3, 8], and transaction management [11, 12].

There are also considerable research efforts devoted to the topic of interconnecting heterogeneous database systems [9, 10, 14, 17, 26]. The main areas of research in DDBMS are File Allocation, Catalog Management, Query Processing, Transaction Management, and Heterogeneous DDBMS. However, our research interest is concentrated mainly on Transaction Management.

7.1.3. Plan of the Report

The rest of this report gives a brief description of a model for distributed transaction processing; as well as its implementation scheme. The primary emphasis of this research was on reliability issues; namely, recovery. Therefore, the issues relating to concurrency control in the proposed scheme are not treated in any detail. However, locking was used as the strategy for concurrency control for the sake of completeness of the model and for the performance analysis.

The next section describes the overall structure of the model and describes some of its implementation details. Section 7.3 presents the conclusions of the qualitative performance prediction of the system. Section 7.4 provides some general information about the implementation of the system. Section 7.5 concludes this report and cites some future research directions. It identifies some possible research extensions to this work in particular and to the area of transaction processing in general.

7.2. The Distributed Transaction Processing Model

The wide-spread use of distributed systems and the novel characteristics of transactions have suggested the extension of the transaction concept to the distributed environment. Transactions that require service at more than one site are called distributed transactions and should still preserve the same transaction properties as the centralized ones. However, the decentralization of transactions raises new issues that are not present in a centralized system. Among these are communication failures and distributed transaction commitment.

A distributed system consists of a computer network in which hosts cooperate towards a goal. This computer network consists of autonomous computers that communicate via a communication network. Message exchanges and data transfers in such a loosely coupled system are prone to more failures than in a tightly coupled system. In addition, they are relatively slow. For real-time (interactive) transaction processing such characteristics are undesirable. One way to improve performance and increase reliability is to minimize the communication overhead of distributed transactions. The transaction processing model presented in the next section reflects such an idea by introducing a new construct, called 'execute', that allows the grouping of several transaction operations under one command.

Our proposed distributed transaction processing model is aimed at minimizing communication in distributed transactions. This communication involves both data transfers and control messages. The proposed scheme, however, addresses mainly the latter. The reason being that the amount of data transfer is dictated more by the application whereas the amount of control messages is dependent mostly on the approach taken to process transactions. The next few sections present a description of the model and a discussion of its

'execute primitive. The underlying system consists of several nodes that can communicate via a network. Each node of the distributed system has one global transaction manager (GTM), one local transaction manager (LTM), one data manager (DM), a storage facility for the database files, and a log management system. The log management system consists of an on--line log storage, a log storage manager (LSM), and an off--line log storage. The logical structure of one node of the system is shown in Figure 7.1. Both the GTM and the LTM have some method of direct access to the on--line storage of the log management system. This is provided so that log information can be forced directly to stable storage. The log files that are no longer needed are transferred periodically to the off--line storage. This operation is controlled by the log storage manager (LSM). The log manager may incorporate some log compression schemes that will speed up the recovery process.

7.2.1. Transaction Processing

The primitives issued by a distributed transaction are `begin_transaction`, `read`, `write`, `execute`, and `commit` or `abort`. They are processed in the model according to the following scheme.

Begin transaction: Upon invocation of this operation, the global transaction manager (GTM) writes a `begin_transaction` record in the log, interprets the request, and decides which nodes are going to be accessed to complete the transaction. Next, it initializes a run--time global table for the transaction. This table has a column for each participating node. The header of the column is the identifier of the called node. Then, it issues a `local_begin_transaction` primitive to all participating nodes. The local transaction managers (LTMs) of these nodes respond by writing a `local_begin_transaction` record in the log and creating and initializing new columns in their run--time local tables. Each of the headers of these columns contains the global transaction number and the identifier of the calling node.

Read(Xitem): The GTM looks in its run_time global tables to find the table of the involved transaction. Next, it looks for the column of the participating node that contains Xitem. If it finds a value for Xitem then it returns it to the calling transaction. If not, it issues a `local_read(Xitem)` primitive to the appropriate node. The LTM of the latter, in turn, issues a `disk_read(Xitem)` primitive to the data

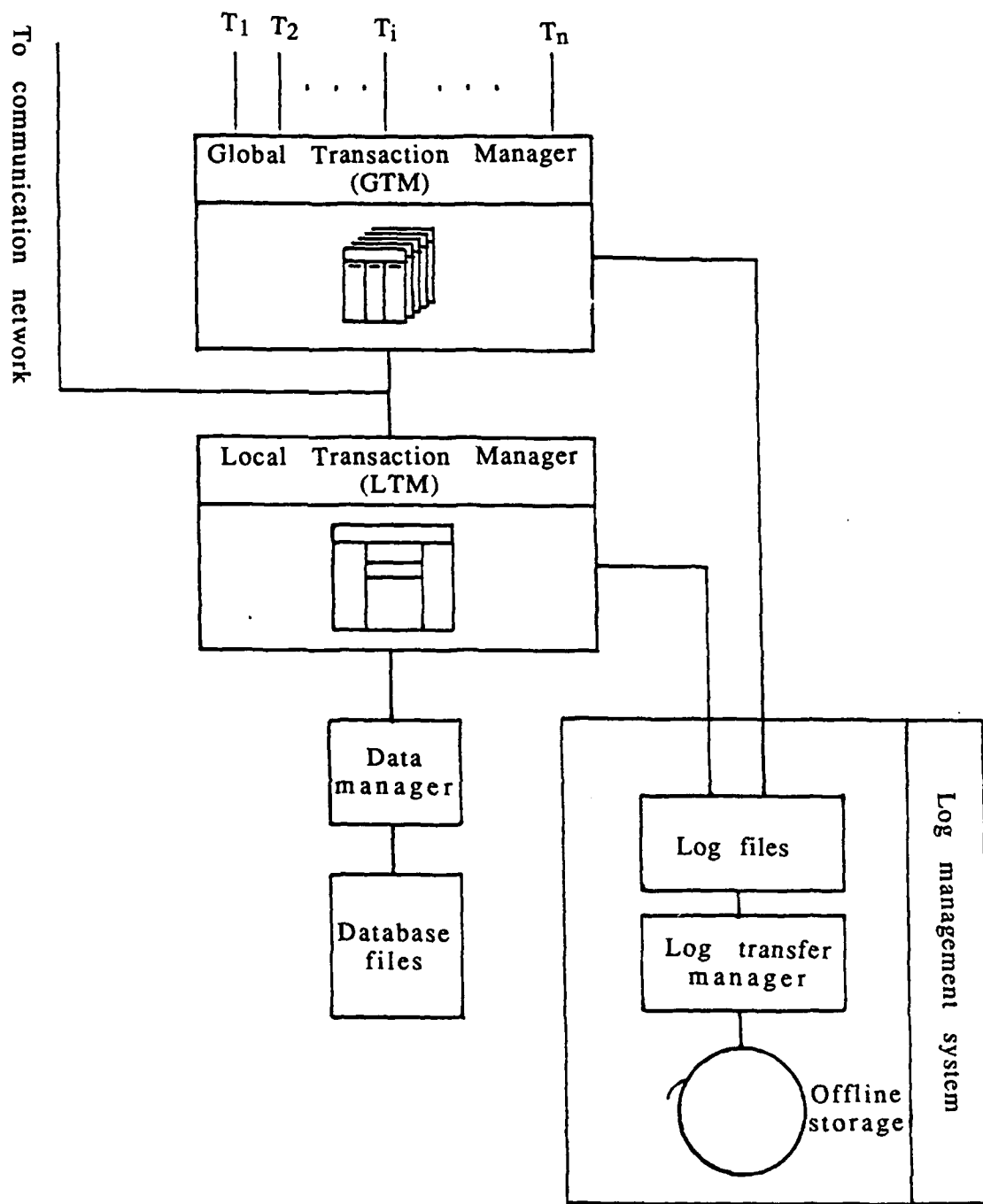


Figure 7.1. One Node of the Distributed Transaction Processing Model

manager of its data storage files. It gets the requested value, adds it as an entry in the corresponding column of the calling global transaction, and sends it to the GTM. The latter adds the information as an entry in the appropriate column of its table and returns the value to the transaction.

Write(Xitem:Xvalue): The GTM checks the transaction's table for an entry of the item. If it finds it, then it updates it and issues a `local_write(Xitem:Xvalue)` to the corresponding LTM and writes a log record of the action. The latter, in turn, updates its copy of the item. Note that if the item has a copy in the global table then it must have a copy in the local table as well. However, If the GTM does not find a copy of the item, then it adds it as a new entry in its table and issues a `local_write(Xitem:Xvalue)` primitive to the corresponding LTM. The LTM, in turn, adds it as a new entry in its run-time table.

Execute(Module_ID,Para1,Para2, ... ,ParaN): Upon receipt of this primitive, the GTM writes the record to the log, records the necessary information in the run-time global table of the transaction, and issues a `local_execute` primitive to the appropriate LTM. Depending on the identification of the module (Module_ID) and the parameters supplied (Para1, Para2, ... , ParaN) the LTM determines the data items to be accessed and the operations to be performed on them, writes a record to the log, performs the operations and records them in the transaction's column, and sends the results, if any, back to the GTM. Note that the type of operations performed and their corresponding logs vary according to the module executed and, possibly, the values of some data. The commitment of the operations of an executed module is fully taken care of by the LTM. Executing a module might require reading and writing data items. These data items are all recorded in the transaction's column, which resides at the local site, but not in its run-time global table, which resides at the site of the GTM. The only information kept at the global table is that of the operation's Module_ID and its parameters Para1, Para2, ... , ParaN.

Commit: Upon receipt of this primitive, the GTM starts the atomic commitment part of the process using the 2-phase-commit protocol. It writes a `Prepare_for_commit` record in the log, sends the message to all participating LTMs, activates a timeout device, and waits for a response from each one of them. The LTMs in turn reply by writing a `ready` or `not_ready` record in the log and

sending the message to the GTM. If a LTM sends a ready message then it must ensure that it is capable of committing its part of the transaction even in the presence of failures. If all of the LTMs respond positively, then the GTM writes a `global_commit` record in the log and issues the primitive to all of the LTMs. Every LTM, in turn, commits its part of the transaction, writes a `final_commit` record in the log, sends the message to the GTM, and removes the transaction's entries from its tables. Upon receipt of the `final_commit` messages from all the participants, the GTM writes a `global_final_commit` record to the log, deletes the run-time global table of the transaction, and informs the end user. If any participant responds with a `not_ready` message, then the GTM must abort the whole transaction.

Abort: To abort a transaction, the GTM writes a `global_abort` record in the log, issues the `global_abort` primitive to all the participating LTMs, and activates a timeout device. The LTMs, in turn, write `local_abort` records in the log, abort their parts of the global transaction, remove the corresponding columns from the tables, and send `final_abort` messages to the GTM. Upon receipt of these messages, the GTM writes a `global_final_abort` record in the log, deletes the run-time global table of the transaction, and warns the end user. However, if the timeout period expires and the GTM does not receive responses from all of the participants, then it writes a timeout record in the log and proceeds as in the previous case.

7.2.2. The 'Execute Construct

Transactions submitted to a particular site are serviced by the GTM and LTM of that site. However, when a transaction requires accessing items at remote sites, the GTM forwards the requests on behalf of the transaction directly to the LTMs at the corresponding nodes. The LTMs are capable of performing several operations on data stored at their sites on behalf of the GTMs. A requested operation could be simple or compound. A simple operation is either a read or a write. A compound operation consists of a collection of read and write operations and is initiated by an `'execute` call.

The `'execute` construct is a one-level macro that translates into the basic `read` and `write` primitives at the local transaction manager's level and is issued by the global transaction managers. This construct enables the programmer to use fewer communications to execute many commands that would otherwise cause heavy traffic due to both data transfers and control-message exchanges.

Suppose, for example, that a global transaction needs to perform N read and write operations at one particular remote site. To initiate one operation, we need a `local_read` or `local_write` message from the GTM to the LTM. In order to initiate the N operations we need N messages across the communication network. If the programmer can group all of these operations under one module name then it can be initiated by only one network message from the GTM to the LTM. This represents a significant gain and the advantage becomes very clear when the communication across the network is expensive, unreliable, or has considerably long delays.

Another advantage of using the 'execute construct is to be able to some processing and subtransaction control locally (at the remote site). This will allow better performance through the overlap of the initiation and execution of many operations by sending only a few messages across the communication network.

The 'execute construct is basically a compound operation that utilizes the proposed architecture. The implementation of the 'execute construct, which belongs to the Atomic Actions Software layer, was built on top of a Remote Procedure Call facility. The hierarchy of software interfaces used in the implementation is shown in Figure 7.2 [25].

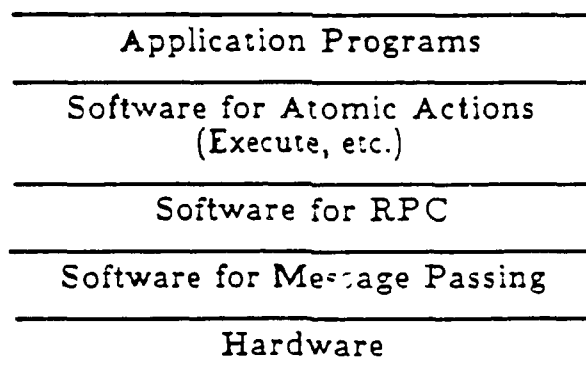


Figure 7.2. Hierarchy of Software Interfaces

7.2.3 Proposed Model vs. The TM/DM Model

The operations requested by transactions fall into one of two categories; process management or data management requests. The `begin_transaction`, `execute`, and `commit` primitives are process management requests whereas the read and write primitives are data management requests. In

the light of this classification of requested operations we compare the proposed model to the TM/DM one [4, 5].

In the TM/DM model, a transaction is represented by one particular process running at one site and controlling the whole transaction. Therefore, the control of the processing of distributed transactions is totally centralized. In the proposed model, however, a distributed transaction is represented by a cooperation of agents. The agents are the LTMs that reside at the different participating sites. The processing of the subtransactions of a global transaction are controlled by the local sites and thus making the control of processing a distributed operation. Figure 7.3 shows the logical structure of the proposed model.

In the TM/DM model, the transaction manager that initiates the distributed transaction is the only one that handles process management requests. The other participating sites handle, exclusively, data management requests. In the proposed model, on the other hand, the local sites handle process management as well as data management requests. This capability reduces to a great extent the amount of data and control messages transferred back and forth between the initiating site (GTM) and the participating sites (LTMs).

7.2.4. The Concurrency Controller

Concurrency control is handled by the data manager (DM) of the site where the requested data resides. For the time being, we are assuming that there is no replication of data. The locking methods of concurrency control resolve conflicts as soon as they arise by blocking the conflicting transactions. The optimistic methods, on the other hand, allow transactions to run to completion before testing for conflicts. Conflicts are then resolved by aborting the conflicting transactions. The choice for the appropriate concurrency control method is very dependent on the application at hand. Several studies however showed that in most applications the concurrency controllers that use locking perform better than those using optimistic methods. Locking induces more rejections while optimistic approaches induce more aborts.

In our model we use locking as the method for concurrency control. Transactions do not need to explicitly request locks. Appropriate locks are automatically granted for all read and write operations and released at commit time. Implementation issues such as the method of setting and releasing locks,

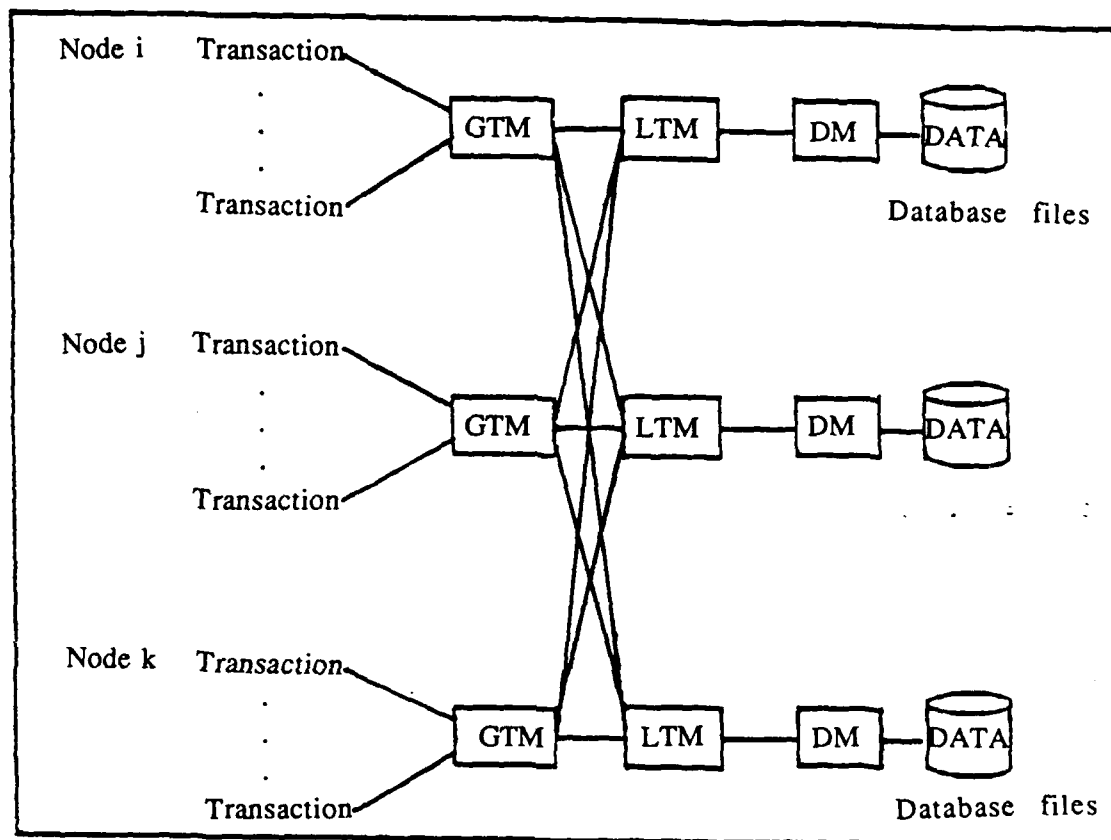


Figure 7.3. Logical Structure of the Proposed Model

the conversion of lock modes, the granularity of accessed data, the scheduling of wait lists, and the handling of starvation and deadlocks are beyond the scope of this report.

7.2.5 The Recovery Mechanism

Recovery deals to a large extent with the problem of preserving transaction atomicity in the presence of failures [11, 12, 15, 16, 27]. In distributed systems, the implementation of atomic actions requires the provision of a special protocol for atomic commitment among sites (global) as well as some recovery capability at every participating site (local).

In the proposed model, atomic commitment is based on the 2--phase--commit protocol. This protocol is resilient to all failures in which no log information is lost. During the commitment stage, when a LTM answers with a ready message to the global commit call, it takes the responsibility of committing its part of the transaction even in the presence of failures. Therefore, If the global decision is to commit and a local failure occurs, then local recovery must be provided to restore the consistency of the local site. Recovery must then be provided at both the local and global levels. The next few sections discuss recovery at the local level and then extend it to the global level.

Recovery at The Local Level: Recovery at the local level is provided by the local site. In our implementation, when a local failure occurs, the local recovery procedure uses the log records to restore the consistency of the database. The only recoverable failures considered are system failures (loss of volatile storage) and media failures that do not involve log files.

Recovery at the Global Level: Atomicity of the distributed transaction depends on atomicity at both the local and global levels. At the local level, and at each participating site, the LTMs must guarantee that all or none of the actions are performed. At the global level, the GTM must guarantee that all the LTMs take the same decision with respect to the commitment or abortion of their parts of the transaction. The local recovery is handled by local recovery managers and thus we only consider recovery at the global level.

The main global system component that is vulnerable to failures is the communication network that connects the different sites of the distributed system. In terms of the communication facility, we assume that if site X sends a

message to site Y and Y receives it, then it was correct and in the proper sequence with respect to other X--Y messages, and within a reasonable predefined period of time MAX_DELAY. Furthermore, if two sites X and Y cannot communicate, then the network is partitioned into at least two groups, one containing X and the other containing Y. This is based on the assumption that as long as there is a path the communication network is capable of routing messages and delivering them properly.

The failures to be considered are those resulting from lost messages and site crashes. When communication failures occur during transaction processing but before the commit operation, then the transaction is simply aborted. However if all operations are successfully completed and a failure occurs at commit time, then it is handled by the 2-phase-commit protocol. A more detailed discussion of the behavior of the 2PC protocol in the presence of the various failures can be found in [20].

7.3 Summary of the Qualitative Performance Analysis

A qualitative performance analysis of the model was carried out in order to predict the performance of the system. In this analysis it is assumed that all service times are fixed (network of delay stations) and thus there is no queuing. The processing of a successful transaction is composed of three phases: Initialization, execution, and commitment. The analysis derived equations for the service times required for processing the different steps of a distributed transaction and then derived an expression for the total time required to complete a transaction. The analysis assumes no failures. The parameters of interest that describe the model and used in this analysis are:

S	The average number of sites accessed by a distributed transaction.
X	The average number of primitive operations performed by an execute call.
N	The average number of calls made by a distributed transaction.
P	Probability that a primitive operation within the execute module is a read
P _E	Probability that the call is a execute

P_W	Probability that the call is a write
P_R	Probability that the call is a read
N_W	Average number of \$write\$ operations in a distributed transaction.
N_R	Average number of read operations in a distributed transaction.
N_T	Total number of primitive operations in a distributed transaction.
C	Capacity of the communication channel.
L_{msg1}	Length of a short message (one packet).
L_{msg2}	length of a long message.
D_{msg1}	Transmission time for short messages.
D_{msg2}	Transmission time for long messages.
D_{stable}	Time required to update stable storage.
$D_{i/o}$	Time required for an I/O operation.
D_{init}	Time required to initialize for a distributed transaction.
D_{oper}	Time required to process a read or a write operation.
D_{exec}	Average time required to process an execute call.
D_{commit}	Time required to process the commitment phase of a distributed transaction.
D_{trans}	Time required to process a distributed transaction.

Here we omit all derivations and analysis and show only the final curves along with some comments.

Figures 7.4, 7.5, and 7.6 show the results of the qualitative performance analysis study. The figures show how the transaction processing time varies with the various parameters of interest.

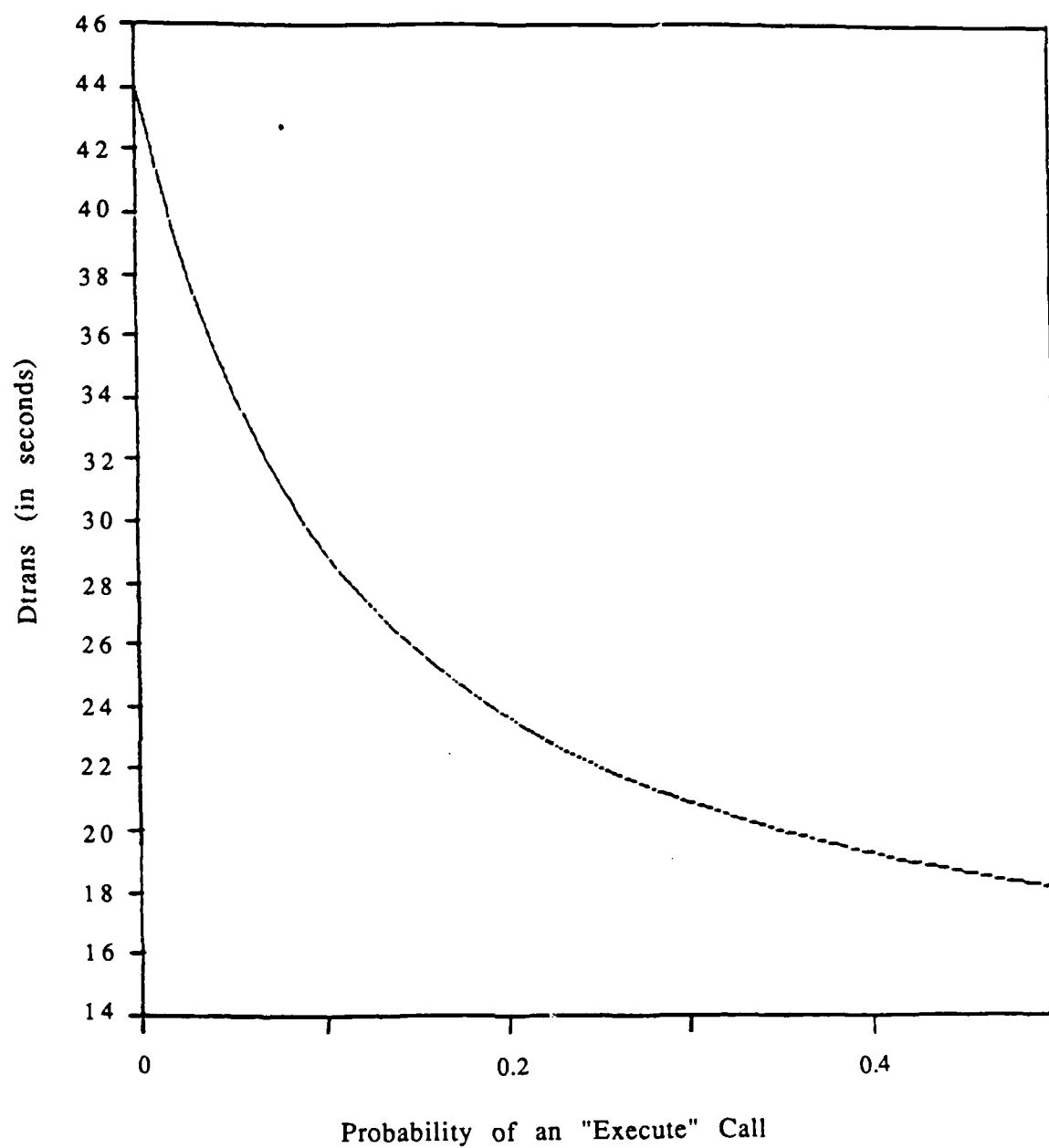


Figure 7.4. D_{trans} versus P_E

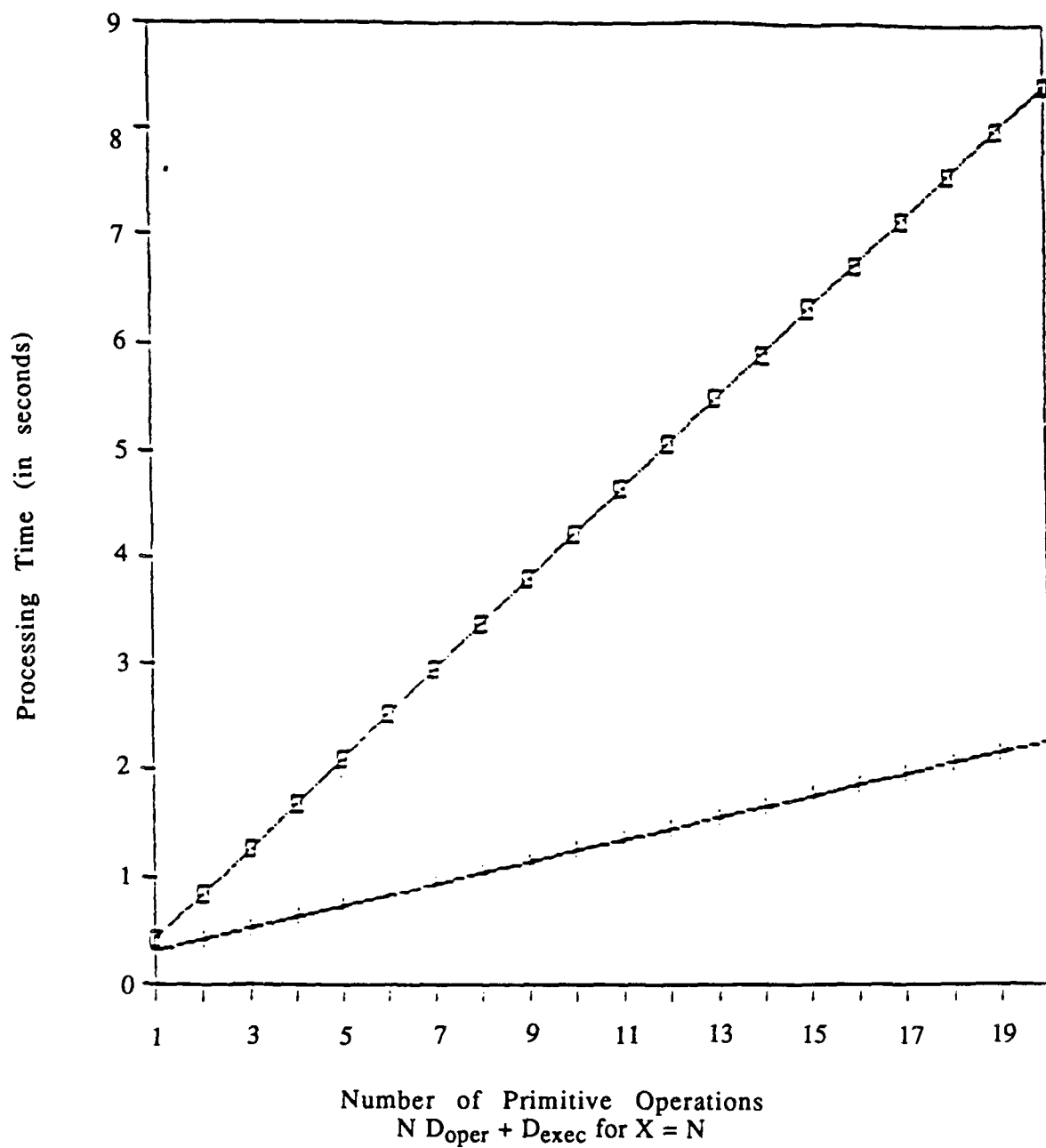


Figure 7.5. $N D_{oper}$ and D_{exec} for $X = N$ versus N

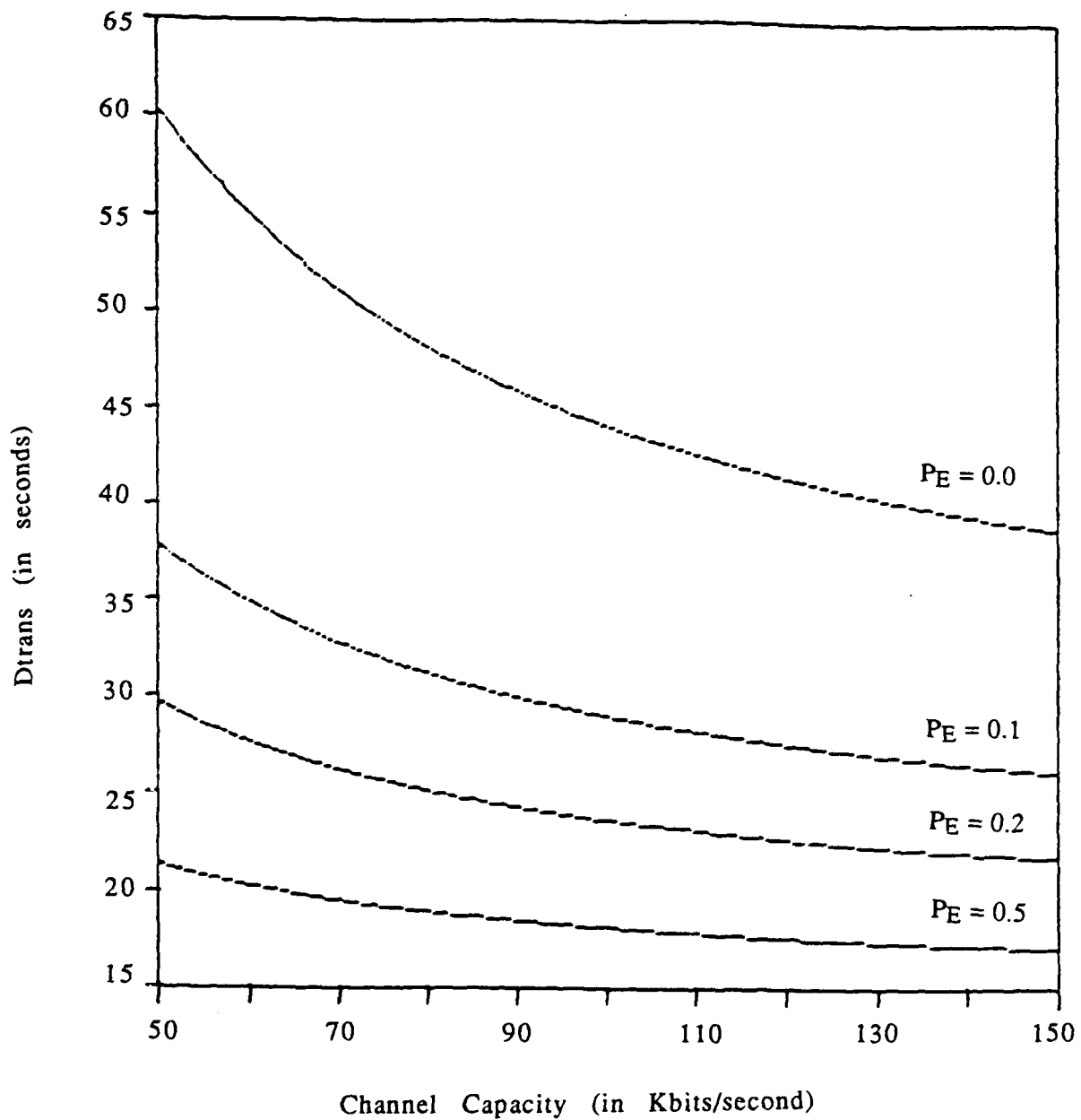


Figure 7.6. D_{trans} versus C for $P_E = 1, 0.1, 0.2$ and 0.5 respectively

Figure 7.4 shows the transaction processing time as a function of the parameter P_E for $C = 100$ Kbits/second. It shows that for a fixed number of primitive operations the transaction processing time decreases exponentially as the frequency of using the execute construct increases. When $P_E = 0$, the transaction processing time is at maximum and the proposed model approximates the TM/DM one. It should be noted that for $P_E = 0$ the proposed model actually takes a longer transaction processing time due to the overhead at the LTM. This overhead consists of the additional local logging that is not present in the TM/DM model.

The steps required for executing an operation in the TM/DM model are the same as those of the proposed one except that there is no local logging. For $P_E = 0$, the processing time for 100 operations is 34.3 seconds in the TM/DM model as compared to 44.131 seconds in the proposed one. However, the proposed model matches the TM/DM one for $P_E = 0.05$ and outperforms it for higher values of P_E .

Figure 7.5 gives a better picture of how the execute construct improves transaction processing time. It shows plots of the quantities $N D_{oper}$ and D_{exec} with $X=N$ as a function of the number of operations N . For a single operation, $N=1$, the two quantities are equal. However, as N increases, the use of the execute construct results in a significantly lower service time due to a lower number of control messages.

Figure 7.6 shows the transaction processing time versus the channel capacity for $P_E = 0, 0.1, 0.2$, and 0.5 , respectively. It can be seen from the figure that as the communication delays become larger, the transaction system shows better performance for higher values of P_E . Therefore, an implementation that utilizes the execute construct more frequently will perform better in the face of communication delays. In fact, the transaction processing time becomes less and less sensitive to communication delays as P_E increases.

7.4 The Implementation

The design presented in this report was implemented in a UNIX environment on a collection of computers interconnected by a local area network (LAN). We have also implemented a Remote Procedure Call (RPC) facility on top of which we built the communication mechanism. The RPC facility was also used in the implementation of the execute construct; with minor changes in the semantics of the calls.

The system is operational and being tested for some performance data to be compared with the predictions discussed in the previous section. We are running the system in a local area network environment and we intend to test it in a long haul network to see the effect of communication delays on the rate of transaction aborts/restarts caused by timeouts.

The experimental system will also serve as a testbed for implementing and testing other transaction management--related algorithms developed by our research group.

7.5 Conclusions and Future Directions

7.5.1 Conclusions

The ever increasing use of distributed computing and the desire for reliable systems led to the extension of the transaction concept to the distributed environment. The primary aim of this research was to study reliability issues in distributed systems and design a distributed transaction processing model.

In this report, we introduced a model for distributed transaction processing in a Local Area Network environment and discussed the details of its implementation scheme. The issues relating to concurrency control in the proposed scheme were not treated in any detail. However, locking was used as the strategy for concurrency control for the sake of the completeness of the model and for the performance analysis. The locking scheme (2PL) was used in the implementation.

The model presented is aimed at minimizing data transmissions and message exchanges between remotely located sites by using the execute construct. This communication reduction will decrease the probability of failures that are due to communication problems as well as increase parallelism in the execution. This will increase both the reliability and efficiency of the distributed transaction processing system. A performance analysis of the proposed system was carried out and here we showed only some performance curves that summarize the analysis. This analysis showed that the use of the execute construct helps achieve better transaction processing time as well as alleviate the effects of communication delays on performance.

7.5.2 Future Research Directions

The next steps of most concern to this work are the design of a more elaborate concurrency controller and the evaluation of the implementation of the prototype. Also, a more detailed performance analysis that relaxes some of the assumptions made by the presented study will be of interest.

A long term research direction is an investigation of the model's feasibility in heterogeneous database management systems. The use of the execute construct provides a way of implementing predefined procedures on a number of database items. Therefore, in a heterogeneous autonomous environment data (objects) will be shared through predefined procedures and the remote foreign users will not have to know the details of implementation of local functions.

Another potential important research area is that of transaction processing in Federated/Heterogeneous DDBMS or F/HDDDBMS. The subject of F/HDDDBMS is becoming increasingly important. It promises to solve many of the problems of accessibility to multiple heterogeneous database systems under temporary or permanent logical interconnections. The main issue that relates to this work is that of transaction management in non-cooperating environments. This topic has been addressed to a reasonable extent but mainly for queries. The problem of global updates still lags far behind in terms of both research and implementation. More research needs to be done in the areas of concurrency control and recovery in autonomous F/HDDDBMS.

Object-oriented databases seem to make more sense for heterogeneous federated systems and it would be interesting to investigate the suitability of the transaction model presented in this report to such environments. Investigations of this transaction model as applied to federated heterogeneous (object-oriented) systems is under way.

7.6 References

- [1] J.E. AllShin and M.S. McKendry, "Facilities for Supporting Atomicity in Operating Systems," Technical Report GIT-CS-83/1, Georgia Institute of Technology, January 1983.
- [2] P.M.G. Apers, et. al., "Optimization Algorithms for Distributed Queries," IEEE-TSE, SE-9:1, 1983.
- [3] P.A. Bernstein and D.M. Chiu, "Using Semi-joins to Solve Relational Queries," Journal of the ACM, 28:1, 1981.
- [4] P.A. Bernstein and N. Goodman, "Concurrency Control in Distributed Database Systems," ACM Computing Surveys, Vol. 13, No. 2, 1981.
- [5] P.A. Bernstein, V. Hadzilacos, and N. Goodman, "Concurrency Control and Recovery in Database Systems," Addison-Wesley Publishing Company, 1987, p. 20.
- [6] S. Ceri and G. Pelagatti, "Distributed Databases, Principles and Systems," McGraw Hill, 1984, pp. 173-206.
- [7] A.K. Elmagarmid and D.N. Mannai, "A Transaction Processing Model for Fault-Tolerant Distributed Computing," Computer Engineering Technical Report TR-86-010, The Pennsylvania State University, 1986.
- [8] R. Epstein, M. Stonebraker, and E. Wong, "Distributed Query Processing in a Relational Database System," ACM-SIGMOD 1978.
- [9] D.K. Gifford, "Weighted Voting for Replicated Data," Proceedings of the Seventh Symposium on Operating Systems Principles, ACM, N.Y., December 1979.
- [10] V.D. Gligor and G.L. Luckenbaugh, "Interconnecting Heterogeneous Database Management Systems," IEEE Computer, January 1984.
- [11] J.N. Gray, "The Transaction Concept: Virtue and Limitations," Proceedings of the Seventh International Conference on VLDB, Cannes, 1981.
- [12] J.N. Gray, "Notes on Database Operating Systems," Operating Systems: An Advanced Course, Springer-Verlag, 1979, pp. 393-481.
- [13] J.N. Gray et. al., "The Recovery Manager of the System R Database Manager," ACM Computing Surveys, Vol. 13, No. 2, 1981.
- [14] A.A. Helal and A.K. Elmagarmid, "Heterogeneous Database Systems," Computer Engineering Technical Report TR-86-004, The Pennsylvania State University, 1986.
- [15] W.H. Kohler, "A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems," ACM Computing Surveys, Vol. 13, No. 2, 1981.

- [16] B.W. Lampson, "Atomic Transactions in Distributed Systems - Architecture and Implementation: An Advanced Course," B.W. Lampson, M. Paul, and H.J. Seigert (editors), Chapter 11, Springer-Verlag, Germany, 1981, pp. 246-265.
- [17] Y. Leu, A.K. Elmagarmid, and D.N. Mannai, "A Transaction Management Facility for Interbase," Computer Engineering Technical Report, The Pennsylvania State University, 1988.
- [18] B.G. Lindsay, "Object Naming and Catalog Management for a Distributed Database Manager," Proc. 2nd Int. Conf. on Distributed Computing Systems, Paris, 1981.
- [19] B. Liskov, "On Linguistic Support for Distributed Programs," IEEE Transactions on Software Engineering, SE-8, No. 3, May 1982, pp. 203-210.
- [20] D.N. Mannai and A.K. Elmagarmid, "Reducing Communications in Distributed Transactions - A Progress Report," Computer Engineering Technical Report TR-87-021, The Pennsylvania State University, 1987.
- [21] D.N. Mannai and A.K. Elmagarmid, "Design and Implementation of a Distributed Transaction Processing System," COMPCON88, March 1988, pp. 185-188.
- [22] J.E.B. Moss, "Nested Transactions: An Approach to Reliable Distributed Computing," Ph.D. Dissertation, MIT, 1981.
- [23] B.J. Nelson, "Remote Procedure Call," Ph.D. Dissertation, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Technical Report CMU-CS-81-119, Xerox Parc, Technical Report CSL-81-9, April 1982.
- [24] C. Pu and J.D. Noe, "Needed Transactions for General Objects: The Eden Implementation," Technical Report, University of Washington, Dec. 1985.
- [25] S.K. Shrivastava and F. Panzieri, "The Design of a Reliable Remote Procedure Call Mechanism," IEEE Transactions on Computers, Vol. C-31, No. 7, July 1982, pp. 692-697.
- [26] J.M. Smith, et. al., "Multibase - Integrating Heterogeneous Distributed Database System," AFIPS Conference Proceedings, Vol. 50, 1981.
- [27] A.Z. Spector and P.M. Schwarz, "Transactions: A Construct for Reliable Distributed Computing," ACM Operating System Review, Vol. 17, No. 2, 1983.
- [28] M.J. Weinstein, et. al., "Transactions and Synchronization in a Distributed Operating System," In Proceedings of the Tenth Symposium on Operating System Principles," ACM, December 1985, pp. 115-126.

CHAPTER 8

SYSTEM LEVEL ERROR CONTROL IN FILE SYSTEMS

8.1. Introduction

Remote replication of data is important for system fault tolerance, since data destroyed at one site remains available elsewhere. It also reduces communication needs during periods of activity, since data is already available at each site. Having the data replicated has an additional advantage which has not previously received much attention: it provides the potential to quickly detect small errors, inconsistencies and alterations in individual copies of the data. This work is concerned with techniques for realizing such potential. Methods were developed of organizing and checking replicated data files in such a way that errors in one or more of the copies can be readily detected and corrected using a minimum amount of communication. A parity checking structure previously developed by Metzner [1] formed the foundation for this study.

8.2. Summary of Principal Results

Divide the file copies into units, denoted as "pages". A quantity called a signature is derived from each page of data, according to parity rules which differ pseudorandomly for different pages [1]. This is done in the same way for all remote copies. To compare files, form various linear combinations of signatures, and communicate these combinations to other sites. A simple example of such a combination is the vector sum of all page signatures of the file. Communication of this vector sum, which may typically be about 40 to 60 bits, it is sufficient to verify with high confidence whether the copies are identical, even if the individual files contain billions of bits. In order to find the pages where large files disagree, additional signature combinations are sent. Techniques were found for which the amount of communication needed to locate the disagreeing page is remarkably small, and the amount of decoding computation is not excessive.

Initially, the work centered on combinational rules based on the structure of a first-order Reed-Muller code. In this case, for a 2^k page file, $k+1$ signature combinations are formed. The combination rules are described by a matrix. For example, for $k=3$, the matrix is

```
11111111
11110000
11001100
10101010
```

Each row of this matrix represents a signature combination, and each column position represents a page; in particular, the combination is the vector sum of all page signatures for which the row entry is a one. It was found that, using these combinational rules, any one or two disagreeing pages could be immediately identified in a computationally relatively simple manner.

A method was then discovered of immediately identifying larger numbers of disagreeing pages by sending a greater number of combinations. The first discovery was that by using a second order Reed-Muller code, which involves sending $1+k+k(k-1)/2$ combinations for the case of 2^k pages, up to any six disagreeing pages could be identified immediately.

The result discovered by Kapturowski for Reed-Muller codes was then generalized by apply to any parity check code by Metzner, and a joint paper based on these ideas has been submitted to the IEEE Transactions on Information Theory [2]. This paper is included as an appendix to the portion of the final report dealing with this subject.

The general result in [2] was that any parity check code with s check symbols and minimum distance d can be used to derive s signature combinations, by which almost any $d-2$ or fewer disagreeing pages can be located. The fraction of nonlocateable patterns of $d-2$ or fewer pages goes down exponentially with signature size, and thus can easily be made negligible. The computational complexity is roughly proportional to the effort involved in inverting a matrix of dimension equal to the number of disagreeing pages. This result represents a large advance over previous schemes [3,4,5], which permitted location of only one or two disagreeing pages with a single data exchange. Reed-Solomon codes are among the most efficient; for these codes only $t+1$ signature combinations need be communicated to locate t or fewer disagreeing pages. As an illustration, if two 1000-page files differed in 5 pages, the location of the 5 disagreeing pages could be found with great confidence by sending only six signature combinations, totaling about 360 bits communicated.

8.3. Potential Applications

1. Protection is provided against accidental or malevolent alteration of any portion of one of the replicated copies - this can be discovered by periodic communication between sites having copies, and the location of the problem can be quickly pinpointed.

2. It can be a backup for consistency maintenance in systems where replicated copies are updated occasionally. Although several algorithms are available to ensure consistency in updating replicated copies, the proposed technique could provide an occasional overall check whether the copies have truly remained identical.

3. In case of failure, it is useful in system recovery. Both the ability to recover and the speed of recovery can be improved. A group at Princeton University reports a recently built triple modular redundancy data base system for which there are plans [5] to incorporate a file comparison algorithm based on the page signature structure that is being considered here.

4. The technique is useful for continually monitoring for system faults. A computer-controlled military system may spend most of its time relatively inactive; the period of full use of its capabilities may be relatively very short. Thus faults may be more prevalent during the inactive period, and it is very important to have means of maintaining the system in the correct mode despite faults during this long period. Fault checking during this period must be thorough, but is not severely time critical. The error-checking techniques described here are well suited for this purpose, and can continually verify proper operation and storage of information in systems employing replication. Sources of error are located efficiently and with minimum communication between elements.

5. In file transfer, it can be used to prevent the need for retransmitting a whole large file if just a part is received in error. Sending back signature combinations will reveal the erroneous pages, if there are any; then only erroneous pages need be resent.

6. The signature computation for a page, compared to a prior computation for that page, can provide automatically a basic need for memory error detection.

8.4. References

- [1] J.J. Metzner, "A Parity Structure for Large Remotely Replicated Data Files," IEEE Trans. on Computers, Vol. C-32, Aug. 1983, pp. 727-30.
- [2] E.J. Kapturowski and J.J. Metzner, "Error-Location Codes - Error Control Through Comparison of Replicated Files," submitted for publication to IEEE Transactions on Information Theory, based in part on Thesis by E.J. Kapturowski for M.S. in Computer Engineering at Pennsylvania State University, June 1988.
- [3] J.J. Metzner, "Efficient Replicated Remote File Comparison," submitted for publication to IEEE Transactions on Computers.
- [4] W. Fuchs, K. Wu, and J.A. Abraham, "Low-Cost Comparison and Diagnosis of Large Remotely Located Files," 5th Symp. on Reliability in Dist. SW and DB Systems," Jan., 1986, pp. 69-73.
- [5] D. Barbara, H. Garcia-Molina, and B. Fetjoo, "Exploiting Symmetries for Low-Cost Comparison of File Copies," presented at 8th International Conference on Distributed Computing Systems, June 1988.

CHAPTER 9

DEPENDABILITY TOOLS FOR BUTTERFLY AND HYPERCUBE COMPUTERS

9.1. Introduction

A major goal in the design of parallel architectures is to provide high computing power with assured dependability*. High computing power can be provided by exploiting the parallelism in the application algorithms and by mapping these parallel algorithms onto the parallel architectures.

The second requirement, "assured dependability" of parallel architectures, stems from the critical applications in which these machine are used. The performance analysis of the parallel system implicitly assumes that the components of a system are fault free. These results give the so called "ideal" performance of a system. However, in a real situation the components of a system fail at random depending on the failure rates of the components. At the system level, a multiprocessor consists of two subsystems. One subsystem is the computation facility which is provided by processors (nodes) and memories. The second subsystem is the communication network, used to support interprocessor communication. The failure of a processor (node) or a memory unit reduces the hardware resources available on the system. The failure of the interconnection switches or links degrades the communication capability of the network. All these faults affect the dependability and performance of the system to varying degrees. A common approach to improve the fault-tolerance of these parallel systems is to provide graceful degradation as an inherent attribute of a system.

Reliability evaluation of parallel systems has been studied under two different approaches, namely: terminal reliability and task based reliability. Terminal reliability is defined as the probability that at least one communication path exists between a pair of nodes. This may be an oversimplified estimate for parallel systems where a job (task) is executed concurrently over several nodes. The task based reliability, on the other had, assumes that a system remains operational as long as a task can be executed with the available resources on the system. This is a more appropriate measure of reliability in a parallel processing domain.

* Dependability is a generic concept that encompasses reliability, availability, maintainability, and safety as distinct facets of system specification.

There are several automatic program packages such as ARIES, CARE III, HARP, SAVE, and SHARPE available for computing the dependability of complex systems. Exact system reliability can be computed using these models. However, these packages are not general enough to handle complex interconnection structures. For example, none of these packages can generate the Markov chain of n-dimensional hypercube automatically. Some form of input, such as a Fault tree of a Markov Chain, of a system is given to these packages for computing reliability. Unfortunately, there is no technique available to generate either the Fault tree or the Markov Chain of a degradable hypercube.

This report summarizes the research efforts accomplished on the dependability evaluation of Butterfly and Hypercube architectures. The BBN ACI Butterfly system is a commercially available MIN-based multiprocessor. Hypercube multiprocessors are a relatively new entrant in the parallel process arena. Hypercube systems with up to 1024 processor nodes are commercially available. The research results are reported in the following two sections. In the next section, evaluation of some of the existing dependability tools is reported. In Section III, a brief review of the analytic models developed for the task-based* reliability evaluation of Butterfly and hypercube systems are presented. The last section describes the future work.

9.2. Evaluation of Existing Tools

There are a number of existing tools available for computing dependability of redundant systems. Tools such as ARIES, CARE III, HARP, and SHARPE can be used for reliability analysis whereas tools such as HARP, SHARPE and SAVE can be used for both reliability and availability analysis. SHARPE, on the other hand, can be used for performability evaluation. In this section a brief summary of CARE III, HARP, and SHARPE is given. The conclusions regarding the applicability of these models to candidate architectures also apply to other tools not summarized here.

9.2.1. CARE III

CARE III (Computer Aided Reliability Estimation, Version Three) is a program designed to estimate reliability of complex redundant systems. It was

* Task-based evaluation is based on the assumption that a system remains operational as long as a task can be executed on the system.

developed specifically for fault-tolerant avionics systems, CARE III features are summarized below.

Capabilities

- Predict the unreliability (1-reliability) of a system consisting of up to 70 stages with each stage composed of one or more identical modules. Can handle hardware/software faults of various types such as permanent, transient and intermittent.
- User must specify the number of modules in each stage, the minimum number of modules needed in each stage for the system to operate properly, the various combination of stage failures that constitute a system failure and the probability that a specific module from stage i forms a critical pair (system failure) with a specific modules from stage j. Hence, a system tree specification involving the critical pairs must be given as input to the program. The lower level faults in the fault tree specification are stage failures.
- Modules imperfect fault handling (coverage) using Markovian technique. Fault distribution is given by a Weibull function.

Disadvantages

- Can not model availability.
- Fault tree in terms of critical pairs of a MIN-based system or hypercube is very difficult. The number of each critical failure combination can be too large to specify for a medium or large size system. Particularly various combination of switch failures that can lead to system failure in a Butterfly type system is extremely difficult to specify.
- Can not model performance-related dependability.

9.2.2. HARP

HARP (Hybrid Automated Reliability Predictor) is a software package developed at NASA. Its advantages and disadvantages are given below.

Capabilities

- Can compute both reliability and transient availability of computer systems using behavioral decomposition along temporal lines. The overall model is decomposed into fault-occurrence/repair (FORM) and fault/error handling (FEHM) submodules to analyze the fault-occurrence and coverage effects effectively.
- Can handle various types of faults as described in CARE III.
- User must input either the Markov chain of the system or a Petri-net model, which can be converted to Markov chain automatically for computing dependability. The other alternative input can be a fault tree specification of the system.
- Can model systems with sequence dependant failures.
- Gives guaranteed bounds on reliability.
- Weibull distribution for reliability modeling.

Disadvantages

- Cannot compute MTTF or steady state behavior for repairable systems.
- Cannot guarantee the Markov chain automatically. As has been pointed out earlier, generation of the Markov chain is complex for systems like Butterfly or Hypercube. Also, a fault-free specification of the candidate parallel system is not simple. Hence, the difficulty of finding the input model restricts the usefulness of HARP to parallel architectures under consideration.

9.2.3. SHARPE

SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator) is currently under development at Duke University [Shaner 87]. In addition to dependability evaluation, it has the capability to include performance with dependability, such as performability. It's advantages are the following.

Capabilities

- Supports seven model types such as reliability block diagram, fault tree without repeat nodes, acyclic Markov chains and irreducible cyclic Markov chains to be combined hierarchically in a flexible manner.
- Allows to use either combinatorial or Markov/Semi-Markov submodules.
- Uses Symbolic computation.

- Input to the model is in the format of reliability block diagram, fault-tree, or Markov chain.

Disadvantages

- As like HARP, construction of the fault-tree of Markov chain is again the challenging problem. Development of a reliability block diagram is also not simple to model task based evaluation.

9.2.4. ARIES

ARIES was designed to satisfy the requirement of analyzing systems which employ protective redundancy techniques such as static, dynamic, hybrid, and gracefully degrading redundancy. A main objective of ARIES is to provide the tools for life cycle analysis.

Capabilities

- Support life cycle measure. Life cycle evaluation includes mission-time, and dependability measures.
- Uses Markov Models for reliability prediction.
- Interactive system.

Disadvantages

- Only exponential distribution can be used for failure and repair phenomena.
- Only one level of redundancy as opposed to multilevel redundancy.
- Complex system is not allowed. Every component in the subsystem should be connected to each other.
- Inaccuracies.

9.3. Research Accomplishments

9.3.1. Butterfly Dependability

Multiprocessor system using multistage interconnection networks (MINS) have been an active area of research for more than a decade. However, there is little reported literature on the analytical modeling of MIN based multiprocessor

systems, mainly due to the NP-hardness of the network structures. We have developed a reliability model for a class of MIN-based multiprocessors, known as Butterfly system. The basic switching elements of the MIN are 4x4 crossbar switches. The name "Butterfly Network" is used because of its topological similarity with that of a Fast Fourier Transform Butterfly.

9.3.1.1. A Reliability Predictor for Butterfly

This section presents an analytical model that we have developed for the task-based reliability evaluation of Butterfly systems. The modeling approach is based on system decomposition and combinatorial techniques. Since the MIN is composed of 4x4 switches, the sys' size is given by 4^i . The reliability of a 4^i system is obtained from four 4^{i-1} subsystems and the connection pattern between those subsystems. We start with probability expressions for 4 PEs and 4 MMs (4x4) and develop a model for a 16x16 system. Results for a 16x16 system are used for analyzing the 64x64 system, which in turn is used in 256x256 system analysis.

There are two different ways a connected group of i PEs and j MMs can be available on the system. The first is the case where exactly i PEs and j MMs are working, and at least the required number of SEs are perfect for providing connection between any PE and MM. In the second situation, more than the required number of processors and/or memories may be working on the system, but, the total connectivity is $(i \times j)$. This is possible when the number of working switches are just sufficient to provide a connectivity (ixj) .

The model is suitable for the analysis of any system size, such as (16x16), (64x64) or (256x256) node Butterflies. Moreover, this model also considers all possible situations for any system size (ixj) . Analytical results for various size Butterfly Systems have been compared with simulation results to show that they are in close agreement.

While our reliability analysis is presented for unique path MIMD architectures, it can be extended to include extra stage of switches to analyze the Butterfly System exclusively. The model can also be applied to multiprocessors using 2x2 SEs without much difficulty.

9.3.2. Hypercube Dependability

Hypercube architectures have received much attention in recent years as vehicles for concurrent processing. A hypercube, also known as Boolean n -cube,

is an n -dimensional multicomputer network. Each of the 2^n vertices of the hypercube is a computer (node) connected directly to its n neighboring nodes. The communication among the nodes is based on a message passing protocol. Since a number of common interconnection topologies, such as ring, tree, and mesh, can be embedded in a hypercube, the architecture is suitable for both scientific and general purpose parallel computation.

9.3.2.1. A Reliability Predictor for Hypercube

We have developed an analytical model for the task-based reliability evaluation of hypercube systems. The model is based on the decomposition principle. A large cube is recursively decomposed into smaller cubes until the modeling of the smallest cube, known as the base model, is simple. A 2-cube (4 nodes) or a 3-cube (8 nodes) is used as the base model hypercube. Reliability of the higher dimension cubes are obtained recursively from either of these base models by approximating the connectivity between two lower dimension cubes.

Najjar and Gaudiot have presented another technique for the reliability evaluation of hypercubes with up to 50% system degradation. Under this approach, the system works as long as there are no disconnected groups. This implies that even if a task requirement is satisfied, the system is considered down if there is any disconnected node(s). However, their model would give a conservative (low) estimate of system reliability particularly for small tasks (more than 50% degradation), since it does not consider all possible situations for i connected nodes in an n -cube. Our model is more general in a sense that it can consider any degradation.

The reliability of the hypercube is calculated by finding the probability of j connected working nodes, $P(C_n = j)$. To calculate the probability $P(C_n = j)$ we divide the n -cube into two $(n-1)$ -cubes (groups). There are two situations under which there will be j connected processors in the n -cube. In the first situations, exactly j connected nodes are working in the hypercube, with k nodes in one group and $(j-k)$ nodes in the second group. In the second situation there are more than j nodes working in the hypercube, but the actual connectivity is only j . For either of these two cases, there are two possibilities for the nature of the connectivity between the two $(n-1)$ -cubes. Either the k and $(j-k)$ nodes working in the two $(n-1)$ -cubes are all connected in their individual groups and the total connectivity is j , or one of the two groups is not internally connected but the total

connectivity is still j . For the second case, where a total of more than j nodes are working in the two groups, the two working groups may or may not be connected.

These four types of distributions are included in the reliability model to find j connected nodes. If a task needs at least I connected working nodes, the system reliability $R_s(t)$ of an n -cube is computed from: ,

$$R_s(t) = \sum_{j=I}^{2^n} P(C_n=j).$$

We have developed a simulation program to validate our analytical model. Analytical results for various hypercubes match nicely with the simulation results. While the analysis is based on the exponential distribution in this report, it should be noted that the model is also valid for other distributions. For example, general distribution like Weibull can be used by simply changing the individual node reliability notation.

9.3.2.2. An Availability Predictor for Hypercube

Availability modeling of a system with single repair facility is more complex than reliability modeling. This complexity can be attributed to two reasons. Firstly, the generation of the Cyclic Markov Chain (CMC) of a system is required for its availability analysis; this can be quite complex for a large system. The number of states in the model can also be too large to handle easily. The second problem is in estimating the transient or steady-state behavior of the system. Closed-form solutions for transient or steady-state availability of a system are extremely difficult, if not impossible, to derive. Therefore, numerical techniques are used to compute availability.

We have developed an analytical model for the availability evaluation of hypercube systems with a single repair facility. The model uses an approximate technique to compute the transient and steady-state availability of hypercubes of any size. Generation of the MC of a hypercube is not required for this technique. The model first computes the probability of x connected nodes, (for $x \leq N = 2^n$), assuming that all the N nodes in a hypercube are fully connected. This problem reduces to a machine repairman model (with or without imperfect coverage) having N nodes. Probability of a state x at any time t can then be computed easily using any standard availability package. This probability is multiplied by a

branching probability that gives the probability of j connected nodes out of x working nodes in a hypercube. The branching probability in turn is computed using the decomposition technique developed for reliability evaluation. A recursive equation is used to compute this probability of an n -cube from a 2-cube base model.

The complexity of generating the exact Markov chain of a repairable hypercube is avoided by using this approximate technique. Under this approach, the generation of the Markov chain is very simple, and the state space is limited in size to no more than $2N$ for imperfect coverage and $N+1$ for perfect coverage.

The validity of model has been examined by comparing the analytical results with simulation results for different task requirements. We have also verified the validity of the analytical model over a wide range of failure and repair rates. In all the cases, the analytical and simulation results match closely.

9.4. Future Work

It was pointed out in the introduction that there exists no tool that can be used for the exact dependability evaluation of the above systems. As architectures become more complex to achieve high performance, the dependability evaluation tools should be able to handle these machines.

Since there is no generic tool that can be used for these complex systems, models have been developed on a case by case basis. However, these models are still in the developing stage. A number of extensions are possible to improve the capability and accuracy of the models.

Work on Dependability Tools

- a) If the Markov states of a parallel machine can be generated, HARP or CARE III type packages can solve the Markov models to find reliability/availability. Automatic stage generation is possible from the fault-tree or petri-net description of a system. Hence, development of fault-trees for various parallel systems is essential. The state generation tool would be used as a front-end package to HARP or CARE III.
- b) We have analyzed only Butterfly network that uses 4×4 switches. Since a lot of multistage networks have been proposed with 2×2 switches, extension of these models for 2×2 switches is proposed.

- c) The idea of the availability model, developed for Hypercube can also be applied to Butterfly. We plan to develop the availability model for the butterfly after the reliability model is completely tuned.
- d) Current experience with HARP indicates that it cannot handle large Markov states. For example, we have difficulty in solving the fault-tree of a 16-node Butterfly with 50% degradation on HARP. Naturally solution of 64-node or 256-node systems would require more capability. Hence, research must be done to handle the generation and solution of large Markov models.

Work on Performance-related Dependability Tools

While classical dependability measures such as reliability and availability are suitable to evaluate uniprocessor systems, these measures may not be good indicators of parallel system behavior. Dependability measures specify only the operational status of a system at any time t . No performance statistics can be gathered from the reliability or availability study. High performance being the main objective of parallel architectures, performance-related dependability is essential to evaluate these machines.

Performance-related dependability measures are unified techniques to evaluate gracefully degrading systems. Measures of this type require the combined evaluation of both performance and reliability.

A number of performance-related dependability measures such as computation reliability, computation availability, performability, capacity and workload characterization, have been proposed for degradable multiprocessors. However, none of these models have been applied in a real sense to Butterfly or hypercube. We would like to combine performance with the previously developed/proposed dependability tools to characterize these architectures.

CHAPTER 10

ALGORITHM MAPPING ON MULTISTAGE NETWORK PARALLEL COMPUTERS

10.1. Introduction

10.1.1. Objectives

Reconfigurability is an attractive property provided by the multistage interconnection network for adapting processor interconnection patterns to meet the communication requirement. The goal of our research efforts is to exploit this valuable property to minimize interprocessor communication cost for parallel algorithms mapped on the multistage interconnection network. In this study we propose two approaches to achieving this goal: (1) Recast parallel algorithms with dynamic communication structures; (2) Design high-level computation primitives with contention-free communication.

10.1.2. Motivations

Designing parallel algorithms with dynamic communication structures is motivated by the desire of altering algorithm communication structure during execution. It has been found that parallel algorithms may have variable communication structures in different computational phases. The flexibility of changing the network structure to match the algorithm communication structure provides the potential of attaining peak algorithm performance. The multistage interconnection network is one of many reconfigurable machines providing such flexibility. Three algorithm designs will be shown in the subsequent sections for illustrating this design approach. The algorithms are designed on the basis of a multiprocessor system model as shown in Figure 10.1.

In parallel processing applications, there are many primitives frequently used to perform global operations, such as data broadcast, selection, collection, and swapping. If these primitives are based on node-to-node communication, a data broadcast, for example, can be viewed as a sequence of one-to-one, store-and-forward communications between the source node and the destination nodes. It is possible for users to implement these primitives by explicit one-to-one communications coded in the program. However, in the multistage interconnection network environment, the user-implemented primitives may incur serious communication delay due to potential network contention. The

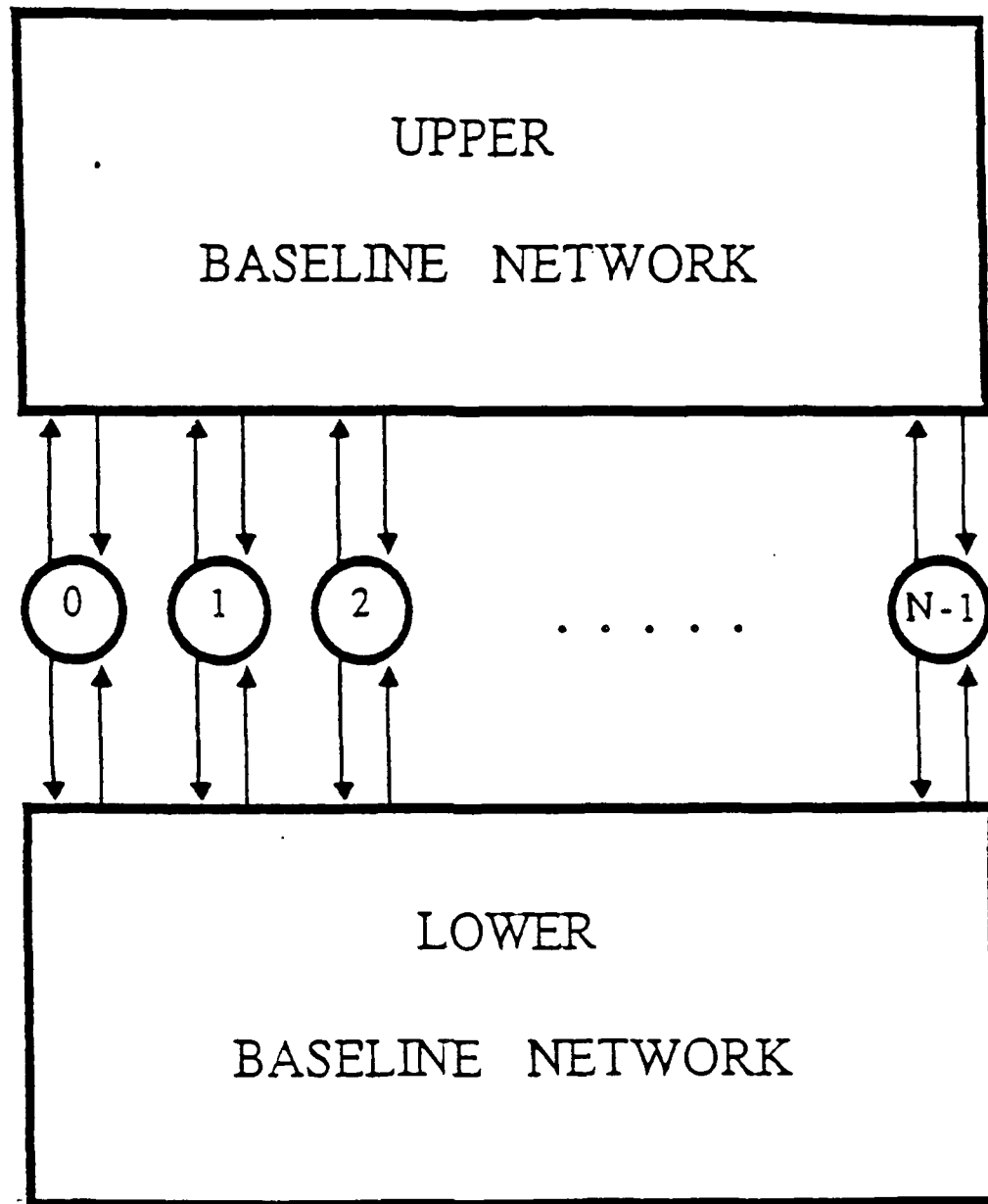


Figure 10.1. The Configurable Multiprocessor System

basic idea of our proposed second approach is to develop contention-free optimal communication structures for these primitives. The use of these structures would relieve the burden of optimizing communication and coupling communication processes from the programmer. Section 5 represents such an effort of mapping a quadtree structure onto the Butterfly network for achieving fast, efficient data broadcast and selection.

10.2. Parallel Conjugate Gradient Algorithm

A parallel algorithm of the conjugate gradient method is presented for solving two-dimensional elliptic partial differential equations [1]. The algorithm is featured by its communication structure, which is dynamically varied in different computational phases to provide efficient interprocess communication. This makes the algorithm different from other works [2-4], where a fixed communication structure has been used throughout the algorithm execution. The target machine of the algorithm is assumed to be a class of reconfigurable multiprocessors [5-8] which can achieve a variety of commonly used topologies to best match communication structures of parallel algorithms.

By the formulation of finite element methods (or finite difference methods), a two-dimensional elliptic partial differential equation is reduced to the solution of a linear algebraic equation: $[A][X] = [b]$, where A is an $n \times n$ symmetric, banded, and positive-definite matrix, X is an $n \times 1$ vector whose values are to be solved, and b is a generating vector from the elliptic partial differential equation. A synchronization mechanism is designed for this parallel algorithm after carefully analyzing the ordering of iterative steps in the conjugate gradient method. To ensure that this synchronized algorithm works correctly and effectively, processes have to synchronize and exchange data with each other. There are a certain number of points where the processes communicate with other processes in this algorithm. These points are called interaction points, which divide the processes into stages. At the end of each stage, a process must communicate with other processes before initiating the next stage of computation.

With this partitioning method, the parallel algorithm is mapped onto a configurable multiprocessor system [5]. The mapping algorithm consists of three horizontal computation stages in each iteration. Each computation stage is then divided into three different computational phases, in which a regular

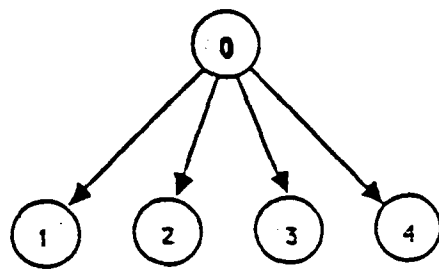
communication structure is provided. Figure 10.2.1 shows two configuration examples achieved by the configurable multiprocessor. It also shows a general communication structure consisting of $2n+1$ processes for solving a problem of size n . The first process is used for interprocess synchronization, while the other $2n$ processes are used for performing the arithmetic computations.

The dynamic communication structures in three computational phases are described as follows. In the first computational phase, a broadcasting tree structure is generated. Then a $2 \times n$ mesh structure follows in the second computational phase. In the third computational phase, a reverse broadcasting tree structure is provided by the configurable multiprocessor.

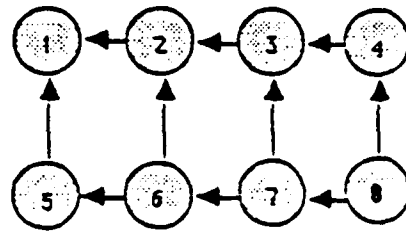
Evaluation of the parallel conjugate gradient algorithm is accomplished by implementing it on a simulator (PAPA), which consists of Pascal-like parallel language and a compiler. The parallel language can provide facilities for creating processes and coupling them through several communication primitives. We use nine processes for solving a linear equation with an $n \times n$ symmetric, banded, and positive definite matrix. The average execution time obtained by using multiple processes is compared with that by using a sequential program running in one process. Based on the simulation results, we observed that the speedup is approximately equal to the number of processes used. Also, the efficiency, which is defined as the ratio of speedup and the number of processes used, of the parallel algorithm is nearly close to an optimal value.

10.3. Parallel QR Algorithm

A parallel QR algorithm with dynamic communication structures is presented for solving the eigenvalue problems of a nonsymmetric matrix [9]. A widely accepted method for computing all the eigenvalues of a nonsymmetric matrix A is to first transform A into an upper Hessenberg form and then to apply iteratively QR algorithm to this upper Hessenberg matrix [10-11]. This is because that the QR algorithm applied directly to a full matrix A (with size $n \times n$) leads to $O(n^4)$ multiplications. If we transform first A into an upper Hessenberg form by a sequence of similarity transformations, it requires $O(n^3)$ multiplications. Then, the application of QR iterations to the Hessenberg form requires only $O(kn^3)$ operations, where k is the average number of iterations for finding one eigenvalue.



Example of a broadcasting tree



Example of a 2×4 mesh

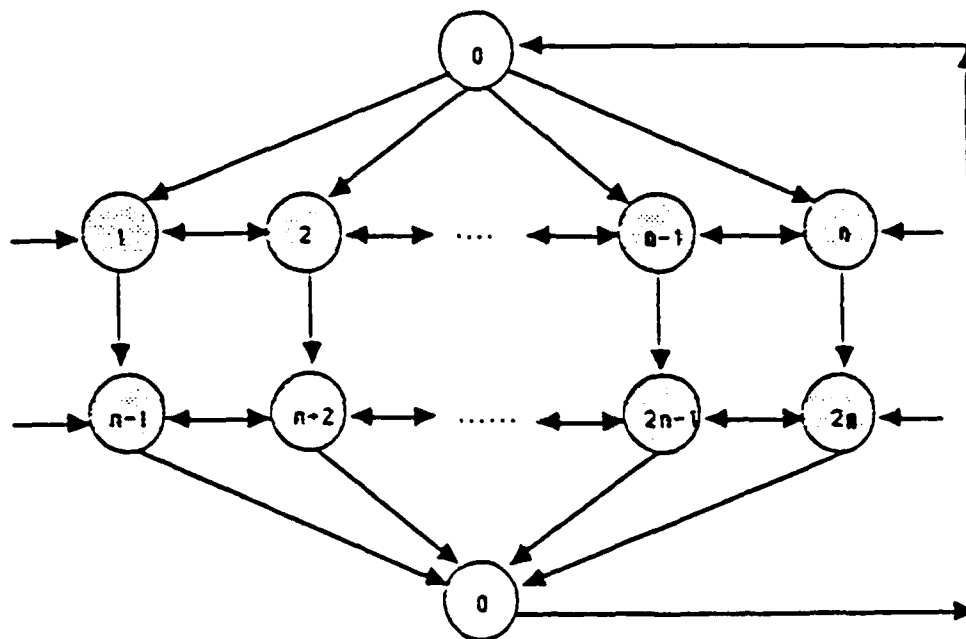


Figure 10.2.1. A General Communication Structure for Solving a Problem of Size n

The algorithm partitioning method is described as follows. First we design a triangular array structure which consists of $\frac{p(p+1)}{2}$ processing elements (PEs) where p is the number of PEs on the edge of the triangular array. An upper Hessenberg matrix is then partitioned into several $m \times m$ blocks. Each partitioned block is assigned to one PE for executing the arithmetic computation. The interprocessor communication requirements therefore need to be satisfied among neighboring PEs. Based on this partitioned method, the grain size ($m \times m$) is determined by the matrix size ($n \times n$) and the number of PEs (p) on the triangular edge. The relation among the three parameters is formulated as $m = \frac{n}{p}$.

The parallel QR algorithm is mapped onto the configurable multiprocessor [5]. There are three computational phases in each QR iteration, and the number of computational phases in each iteration is fixed to three no matter how the matrix size increases. More specifically, the communication structures of the parallel QR algorithm are achievable by the configurable multiprocessor in three different computational phases. In the first computational phase, an upper triangular array structure is generated by the configurable multiprocessor. Then, a $(p-1) \times 2$ grid structure follows in the second computational phase. Then, a $(p-1) \times 2$ grid structure follows in the second computational phase. In the third computational phase, the configurable multiprocessor reconfigures to a lower triangular array structure. The dynamic communication structures make the algorithm provide a very efficient interprocessor communication.

For evaluation the algorithm performance, we first predict it by using a simplified mathematical analysis and then demonstrate the mathematical prediction through an algorithm simulation. Based on the mathematical analysis, we observe that the theoretical speedup has a trend to be an increasing function of the grain size. For the purpose of demonstration, the parallel QR algorithm is implemented and run on the simulator (PAPA). The implementation of the algorithm on PAPA consists of using a fixed number of processing elements (No. of PEs = 10) for solving the eigenvalue problems of nonsymmetric matrices with different size (4×4 , 8×8 , ..., 64×64). Table 10.1 shows the experimental speed up and efficiency as a function of the grain size.

10.4. Parallel Matrix Inversion Algorithm

The problem of inverting matrices is one that frequently occurs in many scientific and engineering applications. It is a problem that in a nature has a

Table 10.1. Experimental Speedup and Efficiency vs. Grain Size

(No. of PEs = 10)

Matrix Size	Grain Size	Speedup	Efficiency
4×4	1×1	2.5	25%
8×8	2×2	3.2	32%
16×16	4×4	3.7	37%
32×32	8×8	4.3	43%
64×64	16×16	5.1	51%

considerably high communication complexity comparable to its computation complexity. Minimization of communication complexity hence has a significant impact on improving the overall algorithm performance. The basic idea behind this work [12] is to make use of reconfigurability of the multiprocessor [5] to distribute computations evenly over processors, and to convey data efficiently to destination processors without incurring significant communication delay.

Previous approaches in matrix inversion are designed for parallel processors with fixed communication structures. For example, LU decomposition on a hexagonal array and cholesky algorithm on a mesh or a cube. The basis of the matrix inversion algorithm to be presented is a compact computational scheme that is originally used for the evaluation of determinants. Our approach is to parallelize and decompose the algorithm into three computational phases. Each phase uses a distinct communication structure.

The most important feature of our algorithm lies in the use of dynamic communication structures. During execution the n^2 processors are dynamically connected into three types of topologies: a Δ -tree, a 2-D mesh bus and a broadcast tree. A broadcasting tree consists of two levels of processors; the root sits at the top while the others are located at the second level. In a Δ -tree of k levels, the number of nodes at each level is incremented steadily by two from top to bottom, and the distance between the root and a terminal node is a constant equal to k . A 2-D mesh bus contains buses in two different dimensions; nodes in the same row or in the same column are all connected by a bus. In order to achieve the three topologies in the configurable multiprocessor, we resolve the potential conflicts among connections through the multistage interconnection networks.

The algorithm performance is analyzed through a simulation experiment. Of interest are speedup estimation of the algorithm and its sensitivity to various system parameters. In order to have insight into performance gain due to

dynamic communication structures, we conduct experiments for comparing the proposed algorithm with a parallel algorithm with an equivalent level of computation parallelism, but with a fixed mesh topology. More specifically, in our experiment we use execution time and speedup as two basic measures to evaluate algorithm performance. Before each experiment we have to set two system parameters: communication cost and reconfiguration cost. The former is an estimate of the penalty that a process has to pay for sending or receiving a message. The latter is an estimate of the response time required by the configurable multiprocessor for adapting its topology.

From the experiments, we observe that the algorithm with dynamic communication structures outperforms the one with a fixed communication structure. We conclude that for the matrix inversion algorithm, the use of dynamic communication structures leads to efficient interprocessor communication and well-balanced computations.

10.5. Parallel Linear Programming Algorithm

Linear programming algorithm is concerned with problems in which linear functions are to be optimized subject to some linear constraints. Studies of the linear programming algorithm indicate that a large amount of time is spent in searching and broadcasting a critical data. These two operations are communication-intensive in nature, and they have a dominant effect on the overall algorithm performance. Thus it is crucial to minimize the interprocessor communication overhead. To attain such a minimization, we propose the parallel linear programming algorithms with two computational phases [13]. The first phase works on searching a minimum value among a series of data located in distinct modules. The second phase endeavors to broadcast the minimum value to other modules.

The target machine is shared-memory, 4×4 switching-based Butterfly Parallel Processor. The core of the Butterfly Parallel Processor is a multistage interconnection network, called the Butterfly Network, through which processor nodes access remote memories in a packet switching manner. The efficient method for mapping the linear programming algorithm is described as follows. A quadtree communication structure and two procedures are presented on the Butterfly Parallel Processor [14]. More specifically, the proposed quadtree communication structure, as shown in Figure 10.5.1, in fact suggests a general

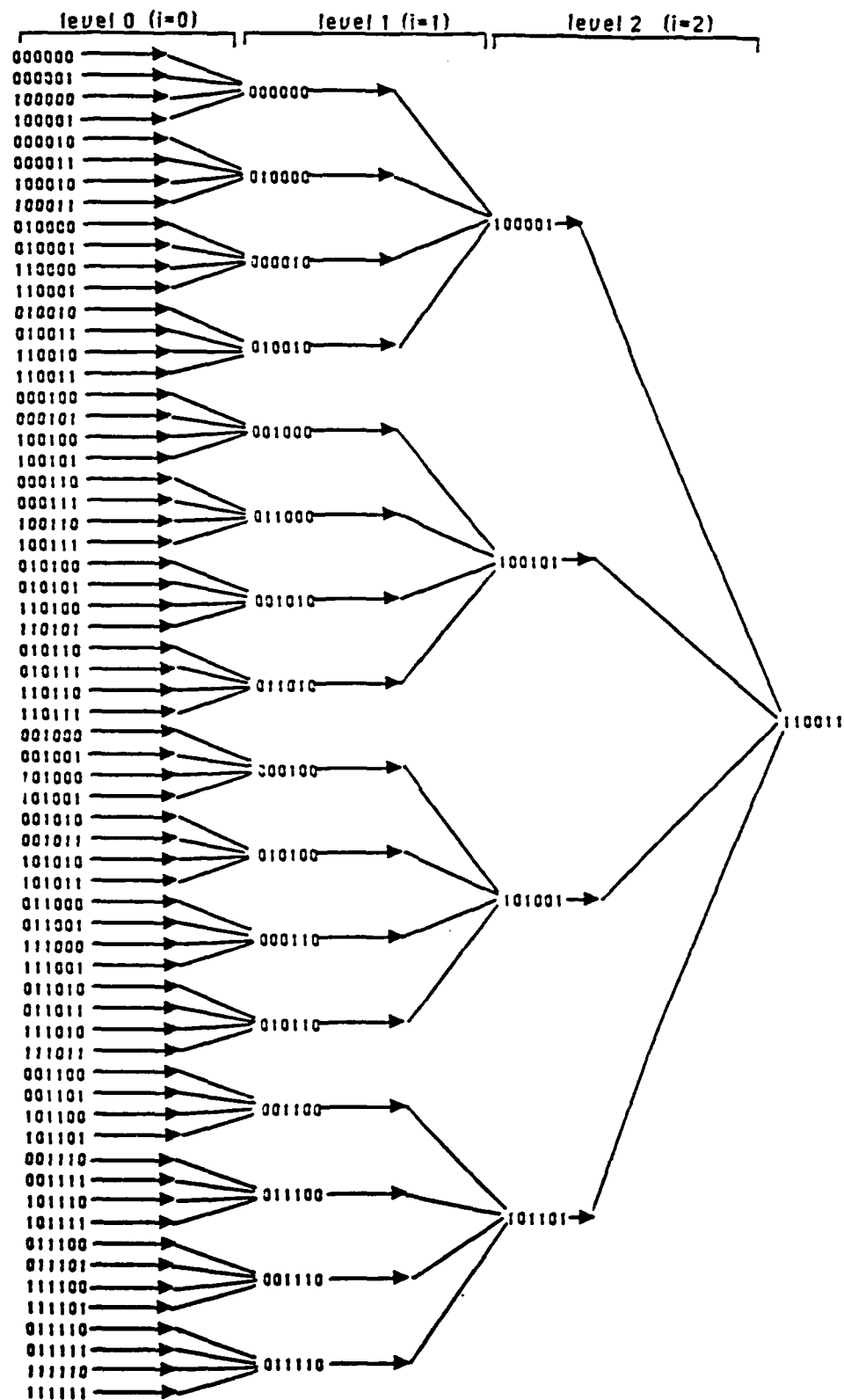


Figure 10.5.1. A Quadtree of 3 tree levels by the Basic Procedure

approach to mapping a class of parallel algorithms with intensive communication requirements for performing two primitive operations - selecting data from many different sources and distributing data from a single source. We show that while performing these two operations through the quadtree structure, the two procedures incur no communication conflicts. By properly merging messages and efficiently replicating data, the parallel linear programming algorithm can accomplish required communications in $O(\log_4 M)$ parallel steps, where M is the size of a Butterfly Parallel Processor. Evaluation of the parallel linear programming algorithm through a mathematical analysis demonstrates that a satisfactory speedup can be obtained by using the quadtree communication structure.

Based on the parallel linear programming algorithm, we present an algorithm extension [15], in which the tree size can be dynamically reduced or enlarged by increasing or decreasing the number of descendent nodes adjacent to each parent node. The tree contraction allows us to balance off computations and communication requirements of algorithms with large computation/communication ratios. Besides, communication links of a contracted tree structure still preserve the conflict-free property of the quadtree structure. For algorithms with small computation/communication ratios, we consider the stretch of the quadtree structure. Without losing the contention-free property, the number of descendent nodes can be reduced to two.

The determination of an optimal number of descendent nodes is dependent on the different computation/communication ratios of various algorithms. By setting up a mathematical model, we analyze the relationship between the optimal number of descendent nodes (ξ) and the computation/communication ratios (π) for a specific network size (m). We see that the quadtree communication structure should be stretched for algorithms with small computation/communication ratios. On the other hand, for algorithms with large computation/communication ratios, we need to contract the quadtree structure to minimize the total algorithm execution time.

10.6. Future Work

We have described two basic approaches to wisely using the reconfigurability of the multistage interconnection network to reduce interprocessor communication cost for parallel algorithms. Our research efforts

have laid down a substantial foundation for future advanced study on the mapping problem. Our next logical step in studying the mapping problem will be the use of network reconfigurability for achieving fault-free communication structures. The main focus is on mapping those useful communication structures we have found in the presence of faulty system components, including processor nodes, network links and switches. We are also interested in investigating the degradation of performance which results from component failures.

10.7. References

- [1] T.L. Sheu, W. Lin and C.R. Das, "An Efficient Parallel Algorithm of Conjugate Gradient Method," in Proc. 1987 Second Int'l Conference on Supercomputing, Volume 2, pp. 488-496.
- [2] P. Concus, G.H. Golub, and D.P. O'Leary, "A Generalized and Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equations," Sparse Matrix Computations, Academic Press, 1976.
- [3] J.K. Reid, "On the Method of Conjugate Gradients for the Solution of Large Sparse Systems of Linear Equations," Proc. Conference on Large Sparse Sets of Linear Equations, Academic Press, New York, 1971.
- [4] J. Reid, "The Use of Conjugate Gradients for Systems of Linear Equations Possessing 'Property A'," SIAM J. Numer. Anal. 9 (1972), pp. 325-332.
- [5] W. Lin and C.L. Wu, "Reconfigurability Procedures for a Polymorphic and Partitionable Multiprocessor," IEEE Transactions on Computers, Oct. 1986, Vol. C-35, Number 10, pp. 910-916.
- [6] S. Yalamanchili and J.K. Aggarwal, "Reconfigurable Strategies for Parallel Architectures," Computer, December 1985, pp. 44-61.
- [7] L. Snyder, "Introduction to the Configurable, Highly Parallel Computer," Computer, Vol. 15, No. 1, Jan. 1982, pp. 47-56.
- [8] C. Davis, S.P. Kartashev and S.I. Kartashev, "Reconfigurable Multicomputer Networks for Very Fast Real-Time Applications," Proc. NCC, 1980, pp. 167-184.
- [9] W. Lin and T.L. Sheu, "A Parallel QU Algorithm with Adaptive Communication Structures," Tech. Report, TR-88-046, Computer Eng., Dept. of Electrical Eng., The Penn State University.
- [10] G.H. Golub and C. Van Loan, Matrix Computations, Johns Hopkins, Baltimore, 1983.
- [11] G.W. Stewart, Introduction to Matrix Computations, Academic Press, 1973.
- [12] W. Lin, M. Thazhuthaveetil and C.R. Das, "A Parallel Matrix Inversion Algorithm with Dynamic Communication Structures," In Proc. of 1988 Int'l Conf. on Supercomputing, Vol. 2, pp. 460-466.
- [13] T.L. Sheu and W. Lin, "Mapping Linear Programming Algorithms onto the Butterfly Parallel Processor," in Proc. of 1988 Int'l Conf. on Supercomputing, Vol. 2, pp. 452-459.
- [14] W. Lin and T.L. Sheu, "A Quadtree Communication Structure for Fast Data Searching and Distribution," in Proc. of Compsac 88, Oct. 1988.

- [15] W. Lin, T.L. Sheu, C.R. Das, C.L. Wu and T.Y. Feng. "Fast Data Selection and Broadcast on the Butterfly Network," in Proc. on Distributed Computing Systems in the '90's, Sept. 1988.

CHAPTER 11

BUTTERFLY PERFORMANCE PREDICTOR

11.1. Introduction

The availability of a variety of commercial multiprocessor computers today makes it difficult to decide on the optimal machine for any specific parallel application area. In addition to the strengths and weaknesses of each candidate machine, the characteristics of programs in the target application domain must be taken into account in making this selection. Unfortunately, neither formal techniques nor software tools are currently available to assist in this decision process. The Butterfly Performance Predictor development effort described here addresses the problem of determining whether a particular class of parallel computer systems is suited to the specific application domains. In particular, the development of software tools for application dependent performance evaluation of the BBN ACI Butterfly Parallel Processor is described.

A variety of parallel processors have appeared on the market in this decade. Most commercially available parallel computer systems can be classified as bus based systems, multi-stage interconnection network (MIN) based systems, or hypercube systems. Bus based multiprocessors consist of processors, memory modules and other devices, connected to each other through a simple computer bus. Examples of this class of system include the Encore Multimax, the Sequent Balance and Symmetry series, and the Synapse N+1 system, with new products announced regularly. These systems are typically restricted in size to a maximum of a few tens of processors due to the performance limitations of current computer buses. In MIN based multiprocessors, the processors, memory modules and other devices are connected through a network of stages of switching elements. The BBN ACI Butterfly system is one commercially available example of a MIN based multiprocessor. The power of the Butterfly MIN makes multiprocessor systems with hundreds of processors cost effective. Hypercube multiprocessors are a relatively new entrant in the parallel processor arena. A hypercube system consists of 2^n processor-memory modules, with each module directly connected to $n-1$ neighbors, forming an n -dimensional cube. Hypercube systems with up to 1024 processor modules are commercially available from Intel Corp, NCUBE Corp, and Ametek.

11.2. Butterfly Performance Predictor Overview

The general operation of the Performance Predictor is illustrated in Figure 11.1. It estimates performance metrics based on two kinds of data: architectural parameters (detailed information about the parallel machine architecture), and algorithm parameters (information about algorithms from the application domain under study). Theoretically, such a performance predictor could be used to study different parallel processor architectures by merely varying the architectural parameters. In practice, it is difficult to conceive of a set of parameters powerful enough to categorize bus-based system, MIN based systems, and hypercube systems in sufficient detail to allow reasonable accuracy of performance prediction. A more conservative design goal was employed in this effort; the architectural parameters were chosen to enable the user to study parallel machines "similar" to the Butterfly.

Figure 11.2 shows the Performance Predictor in more detail; its main component is a **Butterfly Simulator** - a program that simulates program execution on a Butterfly while accumulating performance measures. To drive the simulator under conditions representative of the target application domain, two strategies were considered. In the first strategy, real Butterfly programs are used. This scheme has obvious drawbacks: it requires the simulator to be sophisticated enough to process actual Butterfly machine code and also requires access to programs coded specifically for the Butterfly. A more flexible and user-friendly strategy is to drive the simulator with synthetically generated instruction streams that are representative of the target application domain. The second key component of the Performance Predictor, therefore, is a program (called the **Code Simulator**) that generates these instruction streams.

The work described here was conducted in the period June 1987 - June 1988.

11.3 Butterfly Parallel Processor

The Butterfly multiprocessor system is made up of processor nodes and a Butterfly interconnection network as shown in Figure 11.3. The network is depicted as a cylinder since both its inputs and outputs are processor nodes, unlike a conventional "dance-hall" multiprocessor architecture, which would have processors at one end and memory modules at the other. All of the distributed memory is globally accessible. Remote memory accesses are conducted through the network. Each processor node contains a processor (currently a

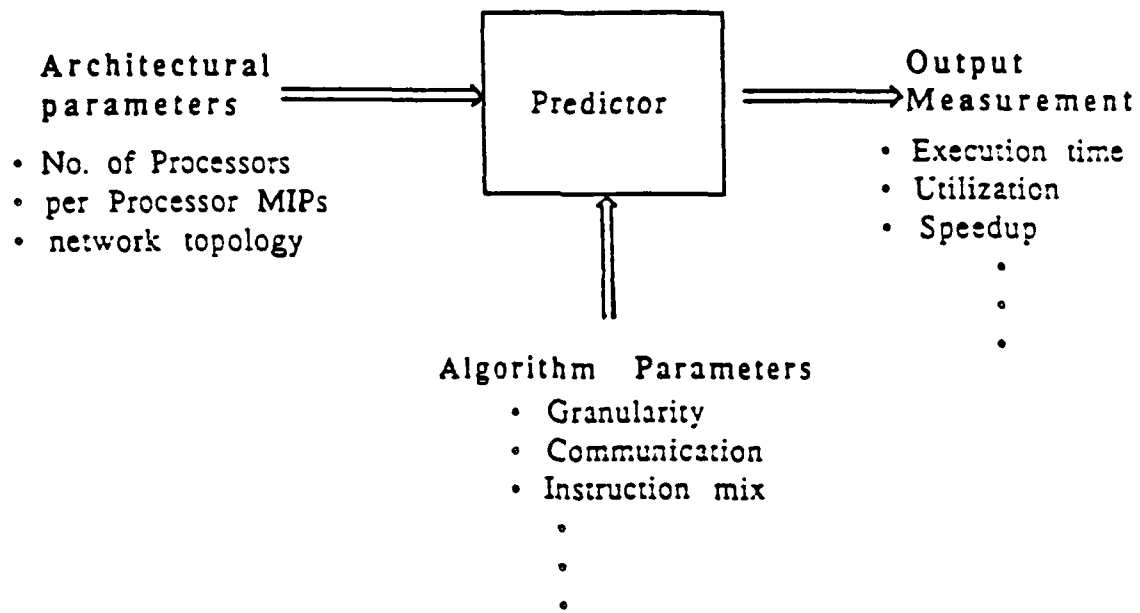


Figure 11.1. Butterfly Performance Predictor

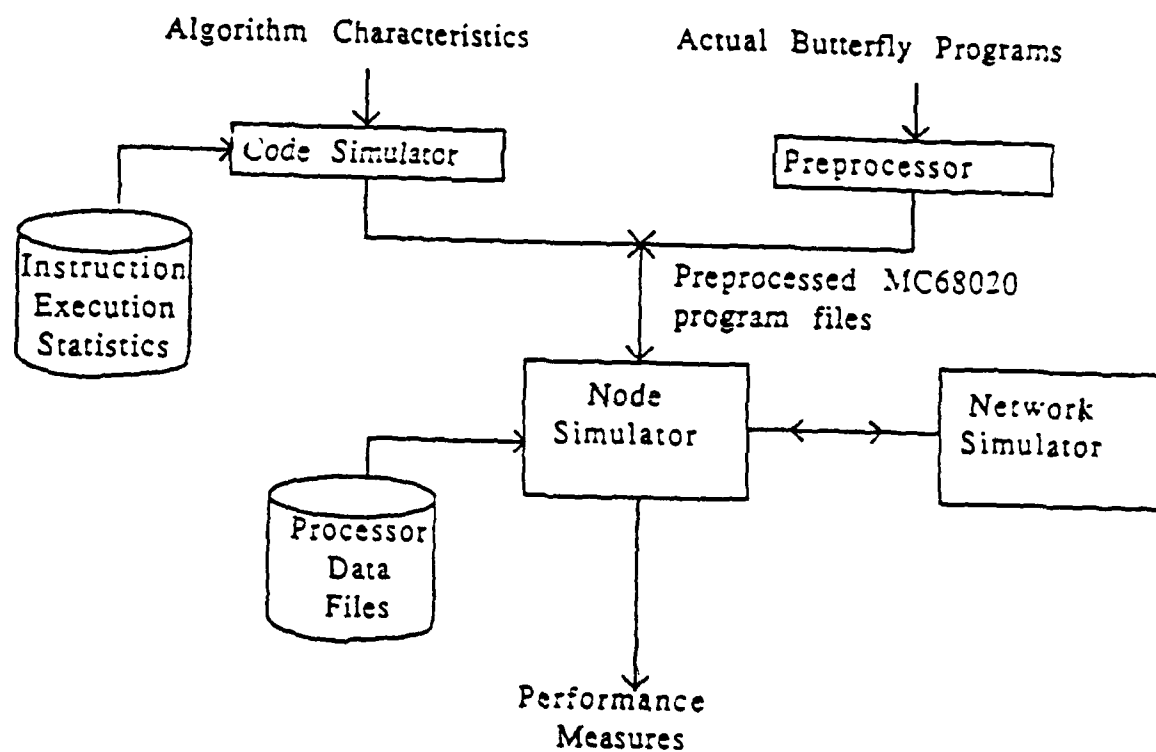


Figure 11.2. Structure of Performance Predictor

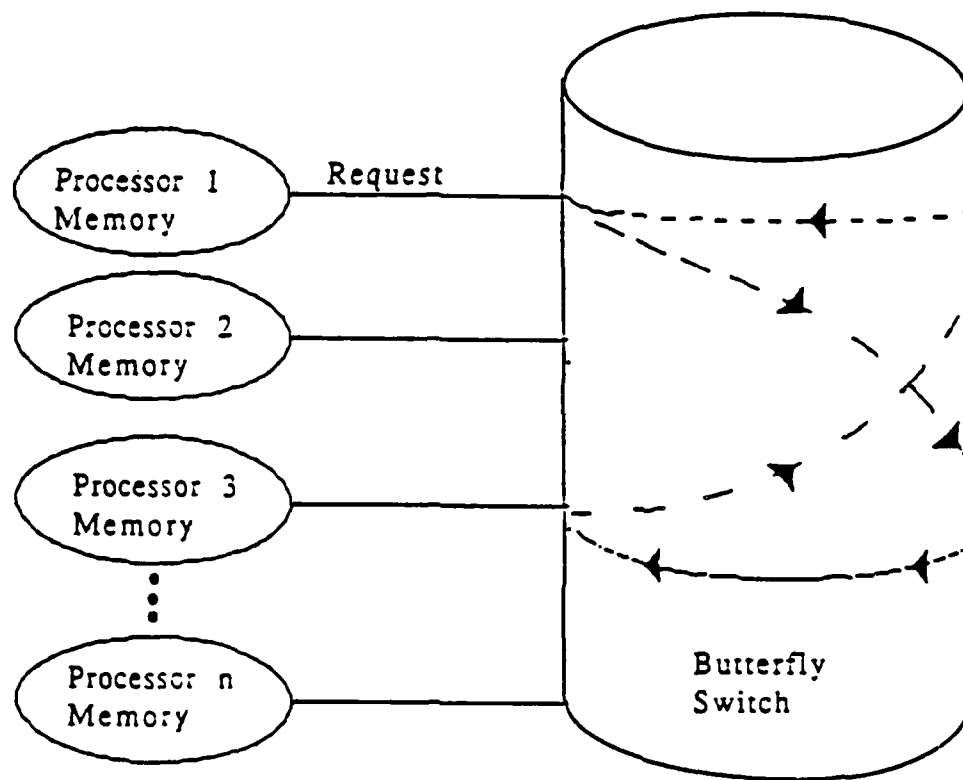


Figure 11.3. The Butterfly Parallel Processor

Motorola 68020), an arithmetic co-processor (MC 68881), 1-4 Megabytes of memory, memory management hardware, and an interface to the network, as illustrated in Figure 11.4.

11.4. Code Simulator

The Performance Predictor, as shown in Figure 11.2, is being designed to accept two forms of input: actual Butterfly program files (after minor preprocessing) and synthetically generated program files representative of algorithms in the application domain of interest. The generation of these synthetic traces is the duty of the Code Simulator. The purpose of the code simulator is to generate representative and meaningful MC68020 code segments that are characteristic of the algorithms of interest. Its operation is based on a set of input parameters:

1. **Granularity:** This reflects the size of the individual parallel tasks comprising the input program. The code generator uses it to decide how large the files of code it generates should be.
2. **Parallelism:** This reflects the number of parallel tasks involved; the code generator uses this parameter to fix the number of MC68020 code files to be created.
3. **Computation/Communication Ratio:** This reflects the extend to which the various tasks comprising the problem communicate. It is used by the code simulator to determine the average number of instructions before a remote memory reference is made.
4. **Number of Communication Partners:** This reflects the communication structure of an algorithm. It is the number of sub-tasks with which a given sub-task is in communication.

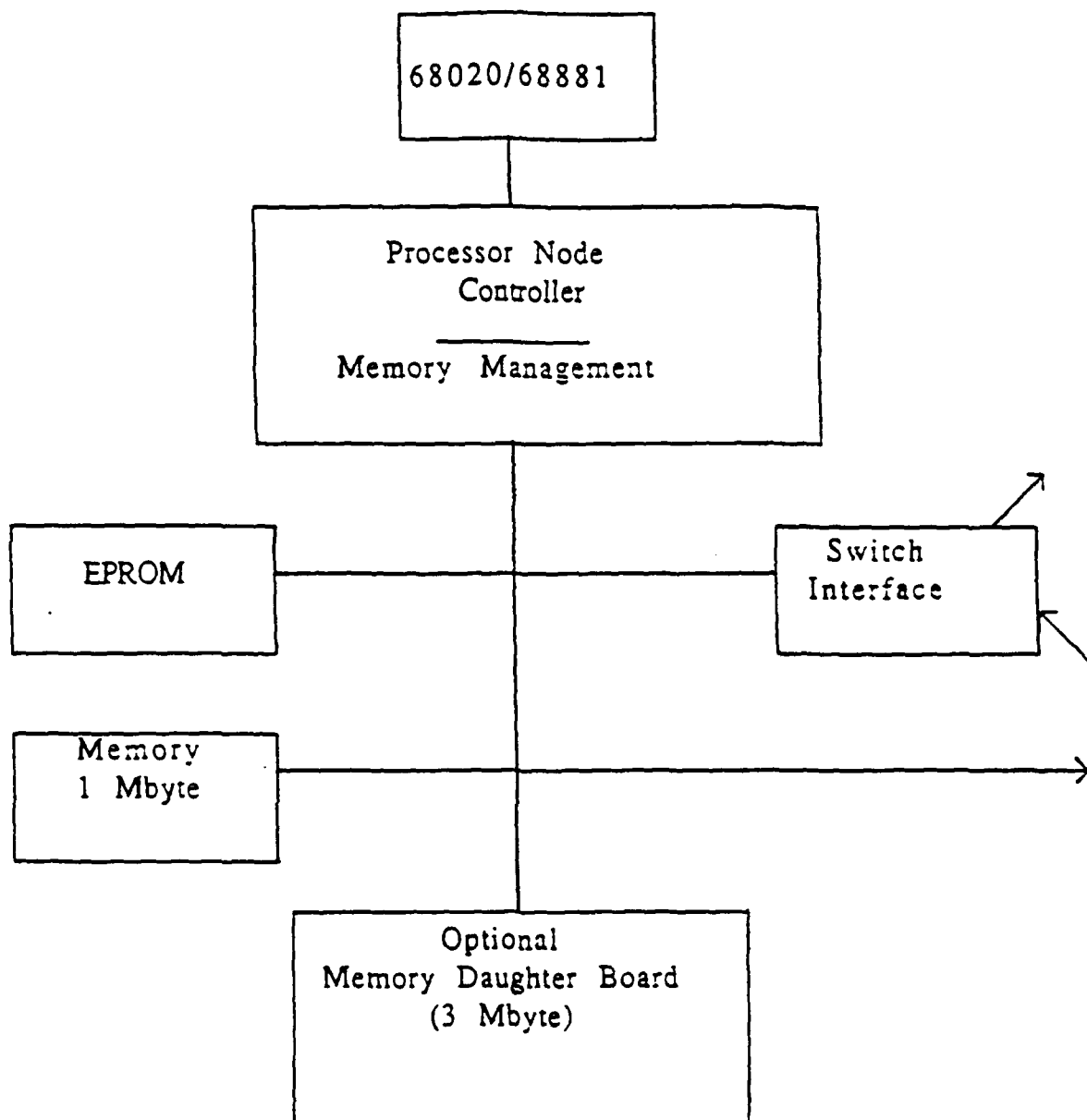


Figure 11.4. A Butterfly Processor-Memory Node

The code simulator is driven by tables of static and dynamic statistics of high level language program contents. These tables were compiled based on a literature survey of instruction execution, instruction transition and operand addressing mode frequencies [Alexander 75, Brookes 82, DePrycker 82, Ditzel 80, Elshoff 76a, Elshoff 76b, Foster 71, Knuth 71, Tanenbaum 78, Wiecek 82]. Since no statistics are available relating directly to the MC68020 instruction set, a set of tables was developed for the MC68020 based on equivalent features reported on for other systems.

The various tables used to drive the code simulator are summarized below:

1. **Instruction Execution Frequency Table:** The information obtained from literature accounted for 64% of instructions executed and included the twenty most frequent instructions. The actual numbers are summarized below. MOVE was reported as the most frequently executed instruction, followed by CMP. The classification of the twenty most frequent instructions executed is not surprising in the light of their function: 61% are integer arithmetic and logical, 25% are related to control and 7% are procedure calls. Within the integer arithmetic and logical class, 59% are moves, 26% are compares, 8% are arithmetic operations, and 7% are converts. Mnemonics for which information was not available in the literature were dealt with by assigning values to them to add up to the remaining 36%. This was done based on information about the percent distributions of arithmetic and relational operators.
2. **Addressing Mode Frequencies Table:** The most frequent addressing modes used in typical programs have also been studied in the literature. They are register (39.9%), byte displacement (14.8%) and indexing mode (7.8%). Instructions having two operands with the form (source destination) accounted for 47.2% of all instructions.
3. **Branch Displacement Table:** Of the instructions executed during typical program runs, 34.7% were branch type instructions. Information on branch displacement frequencies of bit, conditionals and loop branches was used in constructing a table of branch displacements.

4. **Basic Block Table:** It has been observed that on the average a small number of instructions (3-5) are executed before a branch was taken and another instruction sequence path started. This was used to define a basic block: a sequence of instructions starting with a label and terminating in a branch. A basic block table was then constructed. It contained basic block sizes with their expected frequencies of occurrence. The weighted mean of basic block sizes is 3.9.

The generation of MC68020 code files was done using these tables. The tables were summarized in a set of data files containing information on instruction execution frequencies, addressing mode frequencies, branch displacements and basic block sizes.

Based on the granularity parameter, the number of instructions to be generated in a particular file is decided. The file was generated as a series of basic blocks. Labels were calculated using the UNIX random number generator function `drand48()` and the basic block table, and stored in an array. The instructions within a basic block were generated randomly based on the distributions summarized in the instruction execution and addressing mode tables. Separate seeds were used for the function `drand 48()` in deciding the mnemonics and the addressing modes.

In order for the instructions generated to conform with the statistics contained in these tables, checkpoints were introduced: at intervals of 0.1 * number of blocks, the current instruction mix is evaluated. If necessary, the table contents were modified temporarily. This adaptive procedure improves the quality of code generated. A count was maintained for each mnemonic and addressing mode as it was generated. At each checkpoint, the frequency of occurrences of the addressing modes and the opcodes was compared with their respective counts and weighted with a multiplier. The mean square error calculated at the end of each file showed close conformance with the statistics in the tables.

11.5. Butterfly Simulator

The Butterfly Simulator contains two components: a network simulator and a node simulator. The network simulator maintains the status of the Butterfly network while producing timing estimates of how long it takes to traverse the network. The node simulator accepts Butterfly programs as input

and estimates their execution time. It uses the network simulator for timing information related to the Butterfly MIN, and uses a set of files of architectural information to do its own timing estimation. These files contain the "architectural parameters" mentioned earlier and are referred to as the "processor files".

The program execution timing estimates are made at the instruction level. The execution of the program is traced instruction by instruction, and the time taken for each instruction is computed based on timing information obtained from Motorola data books for the MC68020 and the MC68881 and accumulated in the Performance Predictor's processor files: these files contain, for each instruction-addressing mode pair, the time that it takes to execute the instruction on the processor. Figure 11.5 shows sample processor files.

This approach has one serious drawback - it can not take data-dependent (conditional) branches into account in producing its timing estimates. This would have been possible if the Butterfly simulator simulated the actual execution of the input Butterfly programs, which would be slow, costly, and difficult to implement. As an alternative, a probabilistic approach was taken, using statistics from the literature on research into branch prediction [DeRosa 87. Lee 84. McFarling 86. Smith 81] decide whether or not to take conditional branches. These probabilities were user supplied and provide a degree of flexibility to the performance prediction mechanism.

11.6. Conclusion

The code simulator and the Butterfly Simulator have been developed in C on a SUN 3/50 workstation running 4.2 BSD UNIX. Used together, they provide the user with the tools to study the expected performance of classes of algorithms implemented for parallel execution on a Butterfly Parallel Processor.

Processor File Example:

```
16% clock rate in nanoseconds
** % section delimiter
ari% section with mnemonics for operand modes
ard
arid
...
* *
move % section with two operand instructions
add
sub
...
* *
neg % section with one operand instruction
load
...
nop 4 % section with no operand instruction and times
...
* *
bcc 10 15 % section with conditional branch instructions and times
...

* *
bra 10 % section with unconditional branch instructions and times
...
* *
jsr 10 % section with subroutine call instructions and times
...
* *
rtr 5 % section with subrouting return instructions and times
...
* *
frk 20 % section with fork instruction(s) and time(s)
* *
snd 8 % section with send instruction(s) and time(s)
* *
rcv 3 % section with receive instruction(s) and times
EOF
```

Figure 11.5 Sample Processor File

Instruction Time File Example:

```
move % instruction mnemonic
3,4,5;6,7,8; . . . ; 7,8,9;
...
3,4,5;6,7,8; . . . ; 7,8,9;
add
3,4,5;2,4,8; . . . ; 7,8,9;
...
23,24,25;36,37,38; . . . ; 57,58,59;
...
** % section for one operand instructions
neg
3,4,5;
...
5,66,8;
...
EOF
```

Input Program File Example:

```
loop: move ari, arid
add arid, ari
bcc loop
bsr inc
jmp end
inc: add arid, arid
rtr
end: nop
EOP
```

Figure 11.5 (Cont'd) Sample Data Files

11.7. References

- [Alexander 75] Alexander, W.G. and D.B. Wortman, "Static and Dynamic Characteristics of XPL Programs," IEEE Computer, August 1975, pp. 41-46.
- [Brookes 82] Brookes, G.R., and I.R. Wilson, "A Static Analysis of Pascal Program Structure," Software Practice and Experience, Vol. 12, 1982, pp. 959-963.
- [DePrycker 82] DePrycker M., "On the Development of a Measurement System for High Level Language Program Statistics," IEEE Trans. Computers, Vol. C-31, No. 9, 1982, pp. 883-891.
- [DeRosa 87] DeRosa, J.A., and H.M. Levy, "An Evaluation of Branch Architectures," Proc. 14th Annual Int'l. Symp. Computer Architecture, Pittsburgh, PA, 1987.
- [Ditzel 80] Ditzel, D.M., "Program Measurements on a High Level Computer," IEEE Trans. Computers, 1980, pp. 62-72.
- [Elshoff 76a] Elshoff, J.L., "A Numerical Profile of Commercial PL/I Programs," Software Practice and Experience, Vol. 6, 1976, pp. 505-526.
- [Elshoff 76b] Elshoff, J.L., "An Analysis of some Commercial PL/I Programs," IEEE Transactions on Software Engineering, Vol. SE-2, 1976, pp. 113-120.
- [Foster 71] Foster, C., and R. Gonter, "Conditional Interpretation of Operation Codes," IEEE Trans. Computers, Vol. C-20, No. 1, 1971, pp. 108-111.
- [Knuth 71] Knuth, D.E., "An Empirical Study of FORTRAN Programs," Software Practice and Experience, Vol. 1, 1971, pp. 105-133.
- [Lee 84] Lee, J.K.F., and A.J. Smith, "Branch Prediction Strategies and Branch Target Buffer Design," IEEE Computer, January 1984.
- [McFarling 86] McFarling, S., and J. Hennessy, "Reducing the Cost of Branches," Proc. 13th Annual Int'l Symp. Computer Architecture, Tokyo, Japan, 1986.
- [Smith 81] Smith, J.E., "A Study of Branch Prediction Strategies," Proc. 8th Annual Int'l. Symp. Computer Architecture, 1981.
- [Wiecek 82] Wiecek, C.A., "A Case Study of VAX-11 Instruction Set Usage for Compiler Execution", CACM, 1982, pp. 177-184.

DISTRIBUTION LIST

addresses	number of copies
Mark D. Foresti RADC/COTC	15
RADC/DOVL GRIFFISS AFB NY 13441	1
RADC/IMP GRIFFISS AFB NY 13441	2
ADMINISTRATOR DEF TECH INF CTR DTIC-FDA) CAMERON STA BG 5 ALEXANDRIA VA 22304-6145	5
RADC/COTD BLDG 3, ROOM 16 GRIFFISS AFB NY 13441-5700	1
HQ USAF/SCTT Pentagon Wash DC 20330-5190	1
DIRECTOR DMAHTC ATTN: SDSIM Wash DC 20315-0030	1
Director, Info Systems OASD (C3I) Rm 3E187 Pentagon Wash DC 20301-3040	1
Fleet Analysis Center Attn: GIDEP Operations Center Code 30G1 (E. Richards) Corona CA 91720	1

HQ AFSC/XTKT
ANDREWS AFB DC 20334-5000

1

HQ AFSC/XRK
ANDREWS AFB MD 20334-500

1

HQ SAC/SCPT
OFFUTT AFB NE 68113-5001

1

DTESA/RQEE
ATTN: LARRY G. MCMANUS
2501 YALE STREET SE
Airport Plaza, Suite 102
ALBUQUERQUE NM 87106

1

HQ TAC/DRIY
Attn: Mr. Westerman
Langley AFB VA 23665-5001

1

HQ TAC/DRCA
LANGLEY AFB VA 23665-5001

1

ASD/ENEMS
Wright-Patterson AFB OH 45433-6503

2

ASD/AFALC/AXAE
Attn: W. H. Dungey
Wright-Patterson AFB OH 45433-6533

1

AFIT/LDEE
BUILDING 640, AREA B
WRIGHT-PATTERSON AFB OH 45433-6583

1

WRDC/MLPO
WRIGHT-PATTERSON AFB OH 45433-6533

1

AAMRL/HE
WRIGHT-PATTERSON AFB OH 45433-6573

1

Air Force Human Resources Laboratory
Technical Documents Center
AFHRL/LRS-TDC
Wright-Patterson AFB OH 45433

1

2750 ABW/SSLT
Bldg 262
Post 11S
Wright-Patterson AFB OH 454433

1

AUL/LSE
MAXWELL AFB AL 36112-5564

1

Defense Communications Engineering Ctr
Technical Library
1860 Wiehle Avenue
Reston VA 22090-5500

1

COMMAND CONTROL AND COMMUNICATIONS DIV
DEVELOPMENT CENTER
MARINE CORPS DEVELOPMENT & EDUCATION COMMAND
ATTN: CCDE DIOA
QUANTICO VA 22134-5080

2

AFLMC/LGY
ATTN: CH, SYS ENGR DIV
GUNTER AFS AL 36114

1

U.S. Army Strategic Defense Command
Attn: DASD-H-MPL
P.O. Box 1500
Huntsville AL 35807-3801

1

COMMANDING OFFICER
NAVAL AVIONICS CENTER
LIBRARY - D/765
INDIANAPOLIS IN 46219-2189

1

COMMANDING OFFICER
NAVAL TRAINING SYSTEMS CENTER
TECHNICAL INFORMATION CENTER
BUILDING 2068
ORLANDO FL 32813-7100

1

COMMANDER
NAVAL OCEAN SYSTEMS CENTER
ATTN: TECHNICAL LIBRARY, CODE 9642B
SAN DIEGO CA 92152-5000

1

COMMANDER (CODE 3433)
ATTN: TECHNICAL LIBRARY
NAVAL WEAPONS CENTER
CHINA LAKE, CALIFORNIA 93555-6001

1

SUPERINTENDENT (CODE 1424)
NAVLA POST GRADUATE SCHOOL
MONTEREY CA 93943-5000

1

COMMANDING OFFICER
NAVAL RESEARCH LABORATORY
ATTN: CODE 2627
WASHINGTON DC 20375-5000

2

SPACE & NAVAL WARFARE SYSTEMS COMMAND
PMW 153-3DP
ATTN: R. SAVARESE
WASHINGTON DC 20363-5100

1

CDR, U.S. ARMY MISSILE COMMAND
REDSTONE SCIENTIFIC INFORMATION CENTER
ATTN: APSMI-RD-CS-R (DOCUMENTS)
REDSTONE ARSENAL AL 35898-5241

2

Advisory Group on Electron Devices
Hammond John/Technical Info Coordinator
201 Varick Street, Suite 1140
New York NY 10014

2

UNIVERSITY OF CALIFORNIA/LOS ALAMOS
NATIONAL LABORATORY
ATTN: DAN BACA/REPORT LIBRARIAN
P.O. BOX 1663, MS-P364
LOS ALAMOS NM 87545

1

RAND CORPORATION THE/LIBRARY
HELPER DORIS S/HEAD TECH SVCS
P.O. BOX 2138
SANTA MONICA CA 90406-2138

1

AEDC LIBRARY (TECH REPORTS FILE)
MS-100
ARNOLD AFS TN 37389-9998

1

DIRECTOR
NSA/CSS
ATTN: T513/TDL (DAVID MARJARUM)
FORT GEORGE G MEADE MD 20755-6000

1

DIRECTOR
NSA/CSS
ATTN: R24
FORT GEORGE G MEADE MD 20755-6000

1

DIRECTOR
NSA/CSS
ATTN: R21
9800 SAVAGE ROAD
FORT GEORGE G MEASDE MD 20755-6000

1

DIRECTOR
NSA/CSS
ATTN: R5
FORT GEORGE G MEADE MD 20755-6000

1

DIRECTOR
NSA/CSS
ATTN: R8
FORT GEORGE G MEADE MD 20755-6000

1

DIRECTOR
NSA/CSS
ATTN: S21
FORT GEORGE G MEADE MD 20755-6000

1

DIRECTOR
NSA/CSS
ATTN: W3
FORT GEORGE G MEADE MD 20755-6000

1

DOD COMPUTER SECURITY CENTER
ATTN: C4/TIC
9800 SAVAGE ROAD
FORT GEORGE G MEADE MD 20755-6000

1

Attn. Dr. T Feng
Pennsylvania State University
121 Electrical Engineering EAST Building
University Park, PA 16802

5

USAG
Attn: ASH-PCA-CRT
Ft Huachuca AZ 85613-6000

1

JTFPO-TD
Dr. Raymond F. Freeman
Director, Advanced Technology
1500 Planning Research Drive
McLean VA 22102

1

AFEWC/ESRI
SAN ANTONIO TX 78243-5000

3

485 EIG/EIR (DMO)
GRIFFISS AFB NY 13441-6348

ESD/AVS
ATTN: ADV SYS DEV
HANSCOM AFB MA 01731-5000

1

ESD/ICP
HANSCOM AFB MA 01731-5000

1

ESD/AVSE
BLDG 1704
HANSCOM AFB MA 01731-5000

2

HQ ESD SYS-2
HANSCOM AFB MA 01731-5000

1

The Software Engineering Institute
Attn: Major Dan Burton, USAF
Joint Program Office
Carnegie Mellon University
Pittsburgh PA 15213-3890

1